НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ННК "Інститут прикладного системного аналізу"
(повна назва інституту/факультету)

Системного проектування
(повна назва кафедри)

«На правах рукопису»
УДК 004:004.453

«До захисту допущено»
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
"___"_____2018 р.

Магістерська дисертація

зі спеціальності (спеціалізації) 122 – комп'ютерні науки та інформаційні
(код і назва спеціальності)

технології (Системне проектування сервісів)

на тему: Самоналагоджувальна індексна структура бази для діапазонного пошуку за допомогою підходів машинного навчання

Виконав (-ла): студент (-ка) 6 курсу, групи ДА-62м
(шифр групи)

Круш Ігор Володимирович. _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник зав. кафедри, д.т.н., проф., Петренко А.І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант Розробка стартап-проекту д.т.н., проф., Петренко А.І. _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»

Інститут/факультет ННК "Інститут прикладного системного аналізу"
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною (освітньо-науковою) програмою

Спеціальність (спеціалізація) 122 – комп'ютерні науки та інформаційні технології (Системне проектування сервісів)
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
«___»_____2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Крушу Ігорю Володимировичу.
(прізвище, ім'я, по батькові)

1. Тема дисертації Самоналагоджувальна індексна структура бази для діапазного пошуку за допомогою підходів машинного навчання

науковий керівник дисертації Петренко Анатолій Іванович, д.т.н., проф.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Строк подання студентом дисертації _____

3. Об'єкт дослідження повноз'вязні нейронні мережі, метод опорних векторів та методи Байєсівської оптимізації

4. Предмет дослідження (Вихідні дані – для магістерської дисертації за освітньо-професійною програмою) методи машинного навчання у побудові самокерованих індексних структур

5. Перелік завдань, які потрібно розробити
1. Дослідження існуючих рішень, їх аналіз і пошук шляхів покращення.
2. Розробка моделей для повторення результатів з попередніх досліджень та їх покращення

3. Тестування розробленої моделі.
4. Розробка стартап-проекту.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу
презентація на тему «Самоналагоджувальна індексна структура бази для діапазонного пошуку за допомогою підходів машинного навчання»

7. Орієнтовний перелік публікацій:

1. Круш І. В. Адаптивний точковий пошук з використанням методів машинного навчання / Круш І. В., Михалько В. Г. // System analysis and information technology: 20-th International conference SAIT 2018. – 2018. – №20. – С. 177-178.

8. Консультанти розділів дисертації*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розробка стартап-проекту	Петренко А.І., проф.		

9. Дата видачі завдання 01.02.2018

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1	Отримання завдання	01.02.2018	
2	Огляд стану предметної області	15.02.2018	
3	Дослідження теоретичних роботи індексних структур та ідеї самоналагоджувальних індексних структур	05.03.2018	
4	Аналіз існуючих рішень	30.03.2018	
5	Розробка архітектури моделі для діапазонного пошуку	10.04.2018	
6	Реалізація запропонованого методу	17.04.2018	
7	Експеримент з різними типами наборів даних	30.04.2018	
8	Отримання допуску до захисту та подача роботи в ДЕК	10.05.2018	

Студент _____ І. В. Круш
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ А.І. Петренко
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено наукового керівника

РЕФЕРАТ МАГІСТЕРСЬКОЇ ДИСЕРТАЦІЇ

виконаної на тему: Самоналагоджувальна індексна структура бази для діапазонного пошуку за допомогою підходів машинного навчання

студентом: Крушем Ігорем Володимировичем

Загальний обсяг роботи: 103 сторінки, 30 ілюстрацій, 31 таблиця, 1 перелік посилань із 30 найменувань.

Актуальність теми

У зв’язку зі стрімким зростанням кількості даних у мережі Інтернет традиційні підходи до пошуку інформації стають дедалі більш неефективними. Основною їх проблемою лишається те, що вони не враховують реальний розподіл даних та діють з точки зору найгіршого ймовірно розподілу, а їх ефективність вимірюється асимптотичною оцінкою.

Темою дослідження є застосування підходів машинного навчання для побудови нового класу індексних структур для діапазонного пошуку, що будуть враховувати специфіку конкретних даних, на яких будується індекс. Такий підхід дозволяє зменшити використання пам’яті, а також звести час пошуку інформації до константного в незалежності від кількості даних.

Мета та задачі дослідження

Метою даної роботи є пошук шляхів побудови більш ефективних індексних структур за допомогою підходів машинного навчання. Задачею дослідження є реалізація самоналагоджувальної індексної структури для діапазонного пошуку, що враховує розподіл даних та працює за константний час в незалежності від кількості даних та їх природи зростання.

Вирішення поставлених завдань та досягнуті результати

Було перевірено підхід побудови самоналагоджувальних індексних структур за допомогою методів машинного навчання з використанням повновз’язних нейронних мереж та методу опорних векторів з використаням техніки гурту експертів.

Роботу зазначених навчених індексних структур було апробовано на двох типах наборів даних – числових та строкових. Експерименти показали, що цей підхід має

право на життя, і структури доволі успішно вивчили розподіл даних будь-якого типу, однак зі строковими даними результати були кращими.

Об’єкт дослідження

Індексні структури в базах даних

Предмет дослідження

Методи машинного навчання для побудови самоналагоджувальних індексних структур для діапазонного пошуку

Методи дослідження

Досліджується використання нейронних мереж та методу опорних векторів для вирішення задачі вивчення розподілу чисельних та строкових даних. Розроблене рішення використовує сучасні підходи машинного навчання, методи покращення точності у вигляді гурту експертів, а також бібліотеки для тренування і застосування моделей.

Наукова новизна

На відміну від попередніх робіт, які фактично стосувались статичного вибору архітектури, у даному дослідженні було зміщено акцент на побудову спрощених ієрархічних та плоских моделей - використання неглибоких нейронних мереж та методу опорних векторів, а також на використання методів Баєсівської оптимізації для підбору оптимальної по пам’яті та швидкості роботи архітектури моделі в залежності від розподілу реальних даних, що може бути корисним застосуванням для моделей у реальних базах даних.

Практичне значення одержаних результатів

Розроблені індексні структури показують більш ефективне використання пам’яті та приведення часу виконання пошуку до константного, тому, за умови подальшого вдосконалення, вони можуть бути інтегровані у сучасні бази даних.

Крім цього, отримані результати підтверджують загальну ідею доцільності використання машинного навчання для заміни класичних індексних структур.

Апробації результатів дисертації

Результати дослідження оприлюднені на 20-й Міжнародній науково-технічній конференції SAIT 2018.

Публікації

Круш І. В., Михалько В. Г. Адаптивний точковий пошук з використанням методів машинного навчання / Круш І. В., Михалько В. Г. // System analysis and information technology: 20-th International conference SAIT 2018. – 2018. – No20. – С. 177-178.

Ключові слова

Індексні структури, B-tree, машинне навчання, функція розподілу, нейронні мережі, метод опорних векторів, гурт експертів, Байєсівська оптимізація гіперпараметрів

ABSTRACT FOR MASTER'S THESIS

on: Self-tuned database index structures for range queries using machine learning approaches

by: Krush Ihor Volodymyrovych

The thesis contains 103 pages, 30 figures, 31 tables, 30 references.

Relevance

Due to an extreme growth of data on the Internet, traditional approaches for data queries are getting more inefficient. The main issue of those structures is that they do not count real data distribution. They were work as the data has the worst data distribution and we measure their efficiency by asymptotic estimation.

The subject of this work is an application of machine learning techniques for creating a new type of index structures for range queries, which will take into account data patterns. This approach allows reducing memory consumption as well as decrease query time to make it asymptotically constant independently of the amount of data.

Purpose

This work aims to find ways of building more efficient index structures using machine learning approaches. The research objective is to implement a self-tuned index structure for range queries that counts distribution of data and works in constant time.

Results

Index structures developed in this work show more efficient memory usage with constant query time. Therefore, deeper research of the topic should lead to database integration of these index structures.

Object of research

Database index structures

Subject of research

Machine learning techniques for building self-tuned index structures for range queries

Research methods

Neural networks and support vectore machine methods are studied and applied to solve learning data distribution problem for numerical and string data. The developed solution uses modern machine learning methods as well as mixture of models approach to increase accuracy of models built and applied with modern frameworks to models training and inference.

Scientific novelty

Comparing to previous works, which investigated static choise of a model architecture, the accent of current work shifts emphasis on building simplified hierarchical and flat models such as shallow neural networks and support vector machine regression method which are tuned with Bayesian optimization approach aiming to obtain more efficient model architecture in terms of memory usage and speed. The results could be used to build indexes in real-world databases.

Practical value

Developed index structures show more efficient memory usage as well as they reduce asymptotic work time to constant. The result of experiments proves expediency of machine learning usage in building indexes which can replace classical index structures. Therefore, they have a big potential of integrations into modern databases,

Approbation of research results

Research results presented at the 20th International Conference on System Analysis and Information Technology SAIT 2018

Publications

Круш І. В., Михалько В. Г. Адаптивний точковий пошук з використанням методів машинного навчання / Круш І. В., Михалько В. Г. // System analysis and information technology: 20-th International conference SAIT 2018. – 2018. – No20. – С. 177-178.

Keywords

Index structures, B-tree, machine learning, distribution function, neural networks, support vector machine, mixture of experts, Bayesian optimization of hyperparameters.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД - база даних

B-tree – структура збалансоване дерев

GPU – graphics processing unit, графічний процесор, відеокарта

CPU – central processing unit, центральний процесор

GPGPU - general purpose graphics processing unit, неспеціалізовані обчислення на графічних процесорахз

СУБД - система управління базами даних

API – application programming interface, прикладний програмний інтерфейс

CNN – convolutional neural network, згорткова нейронна мережа

ReLU – rectified linear unit, усічене лінійне перетворення

SVM – support vector machine, метод опорних векторів

SGD – stochastic gradient descent, стохастичний градієнтний спуск

ВСТУП

За останнє десятиліття кількість даних у мережі Інтернет стрімко зростає. Корпорації, які володіють соціальними мережами та мереживними додатками усіляко заохочують генерацію контенту користувачами - це дозволяє машинам та їх власникам за допомогою аналітики та підходів штучного інтелекту краще розуміти потреби людей, що ними користуються - наступає ера великих даних. За даними [1], кількість даних, яка генерується у мережі щодня, зростає експоненційно - за прогнозами, у 2021 році кількість трафіку, використаного користувачем за один день, досягне 35 Гб у порівнянні з 13 у 2016. У зв'язку з цим постає проблема збереження та доступу до такого великого масиву даних.

Сьогодні у випадку, коли системі потрібний ефективний у плані швидкості та використання пам'яті доступ до даних, розробники звертаються до індексних структур даних. Вони мають широкий вибір структур, які задовольняють різні потреби в залежності від сформованого шаблону доступу. Наприклад, для діапазонних запитів найкращим варіантом буде B-Tree. Якщо потрібен точковий пошук лише по ключу, важко винайти алгоритм, швидший, ніж Hash-Map. Для виконання перевірки на наявність запису в множині, зазвичай використовують індекс існування, такий як фільтр Блума. Завдяки важливості використання індексів у системах управління базами даних та інших аплікаціях, за останні кілька десятиліть вони були ретельно налаштовані та покращенні, щоб бути ефективними у плані споживання пам'яті, кешу та ресурсів центрального процесора.

Тим не менше, всі ці індекси залишаються структурами даних загального призначення, припускаючи, що розподіл цих даних є найгіршим, і не використовуючи переваг існуючих загальних правил, за якими розподілені реальні дані. Наприклад, якщо метою буде створення високоефективної налаштованої та покращеної системи для зберігання та запитів записів фіксованої довжини з ключами у вигляді неперервних цілих чисел (наприклад, числа від 1 до 100 000 000), то не потрібно буде використовувати звичайний індекс B-Tree над ключами,

оскільки вони самі - ключі - можуть бути використані як зміщення, що робить його доступ асимптотично операцією O (1), а не O (log n) для пошуку будь-якого ключа або початку діапазону ключів. Аналогічним чином, розмір пам'яті під індекс буде зменшений з O (n) до O (1). Звичайно, в більшості реальних випадків розподіл даних не співпадає з певним відомим шаблоном, і також зазвичай немає сенсу розробляти спеціалізований індекс для кожного окремого випадку. Проте, якби ми могли навчити модель, яка відображає шаблон даних, кореляції даних тощо, то можна було б автоматично синтезувати індексну структуру, також відому як навчений індекс, який використовує ці навчені моделі для значного підвищення продуктивності. Останні дослідження [2] показують, що існують реальні випадки, коли навчені індексні структури можуть бути знатно ефективнішими в плані швидкості доступу та пам’яті, ніж класичні індексні структури, зокрема B-tree.

Ще однією важливою причиною потенційно широкого використання навчених індексних структур є прориви у дизайні багатопроцесорних мікросхем, відомих як GPU, ASIC(зокрема, TPU). Відомим є факт, що за останні кілька років фактично зупинився розвиток досліджень у розробці центральних процесорів - закон Мура вже практично не працює, а от для графічних процесорів передбачається зростання в продуктивності приблизно в 1000 разів до 2025 року [3]. І так як більшість алгоритмів машинного навчання найкраще пристосовані саме до роботи на багатопроцесорних схемах (GPU/TPU), що дають всі умови для використання паралельних алгоритмів, найближчі роки їх використання буде все більш і більш поширеним, і, відповідно, слід очікувати покращення у ефективності.

У даній роботі досліджено, наскільки навчені моделі, включаючи нейронні мережі, можуть бути використані для заміни традиційних індексних структур даних як B-Trees. Це може здатися контрінтуїтивним, оскільки машинне навчання не може забезпечити семантичні гарантії, які ми традиційно пов'язуємо з цими алгоритмами, і оскільки найбільш потужні моделі машинного навчання - нейронні мережі - традиційно вважаються дуже дорогими у використанні. Але кожна з цих перешкод настільки ж проблематична, наскільки можуть бути величезними потенційні переваги, які можуть прийти разом з поширенням апаратних тенденцій в GPU і TPU.

У цій роботі розглянуто, як B-tree можуть бути замінені на навчені індекси, а також як можна покращити продуктивність за допомогою останніх напрацювань в розробці апаратного забезпечення.

Метою дипломної роботи є дослідження можливостей використання машинного навчання у побудові самокерованих індексів діапазонного пошуку у системах управління базами даних.

Перш ніж приступити до виконання основного завдання, необхідно провести аналіз існуючих підходів до побудови індексів для діапазонного пошуку у системах управління базами даних. Необхідно розглянути існуючи літературу та дослідження, зрозуміти суть поставлених експериментів.

Після вибору наборів даних, виконати пошук з застосуванням самокерованих індексних структур на основі низки алгоритмів машинного навчання, проаналізувати результати, зробити висновки.

Кінцевою метою роботи є створення тестової системи побудови навчених індексів для чисельних та строкових наборів даних.

1 ОГЛЯД ІСНУЮЧИХ ІНДЕКСНИХ СТРУКТУР І ПІДХОДІВ ДЛЯ ДІАПАЗОННОГО ПОШУКУ

1.1 Діапазонни пошук

Діапазонний пошук - це пошук необхідних даних за заданим діапазоном з урахування порядку, в якому розміщені ці дані. Загалом, найпростішим підходом до пошуку даних є повний їх перебір, але складність такого алгоритму лінійна - O(n), і прості обчислення можуть показати, що він є неоптимальним для дуже великих наборів. Коли необхідно забезпечити швидкий пошук по діапазону для великої кількості інформації, СУБД зазвичай використовують класичні індексні структури, такі як B-tree.

```
SELECT * FROM RELATION WHERE RELATION.key > 4;
SELECT * FROM RELATION WHERE RELATION.key > 1 AND RELATION.key > 10;
```

До прикладу, розглянемо випадок, коли ми маємо індекс, який підтримує запит діапазонів. У цьому випадку дані зберігаються у відсортованому порядку, а індекс створюється для пошуку початкової позиції діапазону в сортованому масиві. Після того, як буде знайдено позицію в початковому діапазоні, база даних може просто отроимати доступ за запитаними даними до кінця діапазону.

1.2 Класичні індексні структури

В системах управління базами даних в більшості випадків використовуються саме Б-дерева (рис. 1.1), які успішно справляються як з діапазонними, так і точковими пошуковими запитами. Також можливо використовувати хеш-індекси для точкових запитів, але сучасні СУБД не радять їх використовувати завдяки гарній оптимізації B-tree індексів, хоч вони в найгіршому випадку виконують пошук

за логарифмічний час O(log(n)), на відміну від константного пошуку за допомогою хеш-індексу O(1).

Загалом, відмінністю між B-tree та хеш-індексами є те, що вони по різному працюють з ключами. У випадку B-tree, основною операцією під час пошуку є порівняння ключів між собою на предмет визначеного порядку - який ключ передує якому. В хеш-таблицях основною операцією є обрахунок хеш-значення ключа за допомогою спеціальної хеш-функції.

При виборі оптимальної структури даних для швидкого пошуку треба також враховувати і тип ключів. Зокрема, для числових ключів обрахування хеш-значення є доволі швидким, в той час як для ключів, що є довгими текстовими рядками, обчислення хеш-значення може бути значно повільнішою операцією.

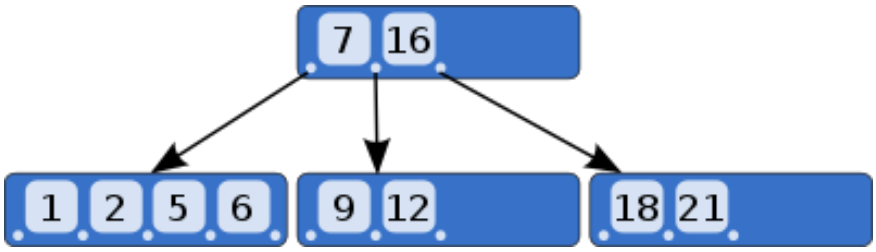

Рисунок 1.1 – схематична структура B-tree

1.3 Класифікація індексів в базах даних

Існує кілька критеріїв, за якими можна розрізняти індекси у базах даних. Архітектурно їх можна поділити на кластерні та некластерні. За типом вони бувають зворотні, розріджені та щільні. [4]

Кластерний індекс тримає ключі у такому ж порядку, як і записи у таблицях, і у однієї таблиці може бути тільки один кластерний індекс (як правило, будується базуючись на первинному ключі таблиці). Використання кластерних індексів може призвести до значного пришвидшення отримання записів з бази даних у випадку, коли фізичний порядок даних, що запитуються, збігається з тим, що зазначено у запиті. Така ситуація поширена, адже дані з жорсткого диску зчитуються блоками, в яких послідовно розміщена велика кількість записів. [4]

Некластерні індекси працюють навіть у випадку, коли дані не відсортовані у якомусь конкретному порядку, адже у структурі індексу зберігаються повні адреси записів. Некластерні індекси зазвичай створюються для колонок, які не виступають в ролі первинних ключів - відповідно, таких індексів може бути декілька для таблиці.

Різниця між щільними і розрідженими індексами полягає в тому, що у щільних індексах зберігаються ключі і адреси для всіх записів, а в розріджених зберігаються адреси на цілі блоки з даними. Розріджені індекси потребують менше пам’яті, але вони можуть бути побудовані тільки для відсортованих даних.

Окремо також потрібно загадати про пошук документів та зворотні індекси (рис. 1.2). Довгий час інформаційно-пошукова спільнота займалася зберіганням документів та ефективним пошуком документів із заданим набором ключових слів. З появою всесвітньої мережі та доцільності зберігання всіх документів в режимі он-лайн, пошук документів за даними ключових слів стало однією з найбільших проблем з базою даних. Хоча існує безліч запитів, які можна використовувати для пошуку відповідних документів, найпростіша та найбільш поширені форми можна побачити в реляційних термінах:

- документ може розглядатися як кортеж у співвідношенні Doc. Це співвідношення має дуже багато атрибутів, кожен відповідає кожному можливим слова в документі. Кожен атрибут є логічним - це слово присутнє в документі, або це не так. Таким чином, схему відношення можна розглядати як **Doc (hasCat, hasDog, ...),** де є дійсно, якщо і тільки якщо документ має слово "кіт" принаймні один раз.
- існує вторинний індекс по кожному з атрибутів Doc. Проте, ми зберігаємо проблеми індексації тих кортежів, для яких значення атрибута має значення FALSE; замість цього, індекс веде нас до документів, для яких слово присутнє. Тобто, індекс містить записи лише для значення пошукової клавіші TRUE.

- замість створення окремого індексу для кожного атрибута (тобто для кожного слова), індекси об'єднуються в один, що називають зворотнім індексом. Це індекс використовує непрямі кошики для ефективності використання пам’яті.

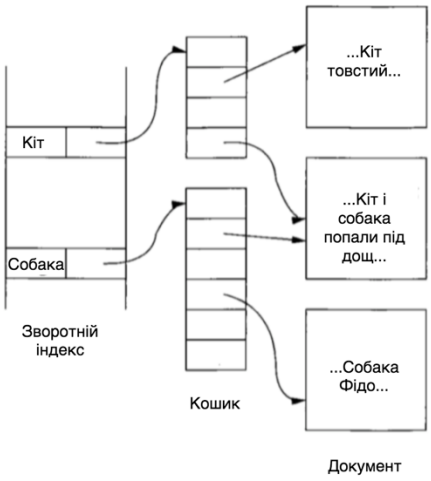

Рисунок 1.2 – схематична структура інвертованого індексу

1.4 B-дерева

В галузі інформатики, B-дерево - це самобалансована деревоподібна структура даних що, що дозволяє організувати сортування, пошук, послідовний доступ, вставки та видалення даних за логарифмічний час.

B-дерево є узагальненням двійкового дерева пошуку, в якому вузол може мати більше двох дітей. На відміну від самобалансованого дерева бінарного пошуку, B-дерево оптимізоване для систем, які читають і записують великі блоки даних. B-дерева є гарним прикладом структури даних для доступу об’єктів із зовнішньої пам'яті. Вони зазвичай використовується в базах даних і файлових системах.

В B-деревах внутрішні (не кінецеві) вузли можуть мати змінну кількість дочірніх вузлів в межах певного заздалегідь визначеного діапазону. Коли дані вставляються або видаляються з вузла, кількість його дочірніх вузлів змінюється. Щоб зберегти заздалегідь визначений діапазон, внутрішні вузли можуть бути об'єднані або розділені. Оскільки допустимий діапазон дочірніх вузлів визначений, B-дерева не потребують повторного балансування так часто, як інші самобалансовані дерева пошуку, але можуть витрачати більше пам’яті, оскільки

вузли зазвичай не повністю заповнені. Нижня та верхня границі кількості дочірніх вузлів, як правило, фіксуються для певної заданої реалізації. Наприклад, у 2-3-B-дереві (часто просто називають деревом 2-3), кожен внутрішній вузол може мати лише 2 або 3 дочірні вузли. Кожен внутрішній вузол B-дерева містить декілька ключів. Ключі виступають як роздільне значення, що утворює два пів-дерева. Наприклад, якщо внутрішній вузол має 3 дочірні вузли (або під-дерева), то він повинен мати 2 ключі: **a1** і **a2**. Усі значення в лівому піддереві будуть меншими за **a1**, всі значення в середньому піддереві будуть між **a1** і **a2**, а всі значення в правому піддереві будуть більшими, ніж **a2**.

Зазвичай кількість ключів змінюється між **d** та **2d**, де **d** містить мінімальну кількість ключів, а **d + 1** - це мінімальний степінь або розгалуження дерева. На практиці, ключі займають найбільше місця в вузлі. Степінь 2 гарантує, що вузли можна розділити або об'єднати. Якщо внутрішній вузол мав **2d** ключів, то додавання нового ключа до цього вузла може бути виконане шляхом розбиття гіпотетичного вузла ключа **2d + 1** на два **d** ключі і переміщаючи ключ, який був би посередині по відношенню до батьківського вузла. Кожен розділений вузол має необхідну мінімальну кількість ключів. Аналогічним чином, якщо внутрішній вузол та його сусіди мають **d** ключів, то ключ може бути видалений з внутрішнього вузла, об'єднавшись зі своїм сусідом. Видалення ключа залишить **d - 1** ключ у внутрішньому вузлі; приєднання до сусіда додасть **d** ключів та ще один ключ, опущенний від батьківського вузла сусіда. Результатом є повністю заповнений вузол з **2d** ключами.

Кількість гілок (або дочірніх вузлів) у вузла буде більшою, ніж кількість ключів, що зберігаються в вузлі. У 2-3 B-дерева внутрішні вузли зберігатимуть або один ключ (з двома дочками) або два ключі (з трьома дочками). Іноді B-дерево описується параметрами **(d + 1)**, **(2d + 1)** або просто найвищим порядком розгалуження **(2d + 1)**.

B-дерево зберігається врівноваженим, вимагаючи, щоб всі кінцеві вузли були на одній глибині. Ця глибина збільшуватиметься повільно, оскільки елементи

додаються до дерева, однак збільшення загальної глибини нечасте, і всі кінцеві вузли стають ще одним вузлом щоразу далі від кореневого вузла.

Розглянемо спосіб доступу до даних, що лежать у B-tree. Для простоти припустимо, що B-дерева мають суттєві переваги перед альтернативними імплементаціями, коли час доступу до даних вузла набагато перевищує час, витрачений на обробку цих даних, оскільки тоді вартість доступу до вузла може бути амортизована протягом декількох операцій в межах вузла. Це зазвичай виникає, коли дані вузла знаходяться в вторинному сховищі, наприклад, на дисках. Максимізуючи кількість ключів у кожному внутрішньому вузлі, висота дерева та кількість дорогих в плані доступу вузлів зменшується. Крім того, перебалансування дерева відбувається рідше. Максимальна кількість дочірніх вузлів залежить від інформації, яка повинна зберігатись для кожного дочірнього вузла, а також від розміру блоку заповненого диска або аналогічного розміру у вторинному сховищі. Хоча 2-3 B-дерева легше пояснити, на практиці B-дерева, що використовують вторинне сховище даних, потребують великої кількості дочірніх вузлів для підвищення продуктивності. [5]

1.5 Паралельний пошук у GPU-базованих базах даних

Як було зауважено, протягом останніх кількох років спостерігається тенденція до збільшення паралельної продуктивності графічних процесорів для обчислювальної техніки загального призначення (GPGPU). Запит великої кількості результатів у великій базі даних є затратною у плані часу та пам’яті задачею, і попередні дослідження показали, що запуск запитів на GPU через CUDA може призвести до значних прискорень роботи системи [6]. Проте ці підходи потребують суттєвих змін традиційної бази даних та базових структур даних. Як відомо, для збереження записів в організованому вигляді у більшості звичайних базах даних використовується дерево B+, яке побудоване як N-арне дерево пошуку з елементами, котрі записані на листах дерева. Хоча операції дерева за своєю суттю є впорядкованими, процес перевірки запиту серед ключів вузла може бути

розпаралелений. Доведено, що можливе досягнення збереження алгоритму дерева B+ водночас із використанням паралельної архітектури GPU для підвищення продуктивності системи. [7]

CUDA - це програмна архітектура, розроблена компанією NVIDIA, яка дозволяє програмістам легко писати паралельні програми, що працюють на графічному процесорі. CUDA використовує ієрархію груп потоків, загальну пам’ять та бар’єрну синхронізацію, доступні через API в C. Кожен потік можна ідентифікувати до 3-х вимірів. Потоки, згруповані разом, утворюють блоки потоку, які запускаються на одному з мультипроцесорів SIMD, називається SM. Блоки потоків відділяють власну пам'ять від глобальної пам'яті, і це розділення пам'яті та блоків потоків є основою для підходу до запуску декількох окремих запитів одночасно. [7]

Плетений паралелізм є загальною технікою, що використовується в обчисленні GPGPU, в якій програміст виконує паралельно декілька незалежних завдань на GPU. Ця парадигма добре підходить для алгоритму обходу B+ Tree та структури CUDA окремих блоків потоку: кожен блок потоку, який працює на одному SM, обробляє окремий запит, тоді як різні запити обробляються паралельно для різних SM, використовуючи різні потокові блоки, що підвищує загальну пропускну спроможність. Для цього потрібна лише одна передача даних для декількох запитів, при цьому повністю використовується паралельний характер графічного процесора. Результати показують, що "batched queries" дають прискорення до 16 раз, включаючи час передачі послідовного виконання набору результатів на процесорі. [7]

1.6 Алгоритм розподіленого пошуку у B-tree

Щоб оптимізувати зберігання дерева для GPU, треба зазначити, що всі сторінки структурно схожі і будуються регулярно.

Нехай **N** позначає порядок дерева або кількість елементів, збережених на одній сторінці. У листі нам потрібні **2N** позицій для ключів і значень плюс лічильник для кількості зайнятих місць і вказівник на наступну сторінка. У вузлах нам потрібні **N** місць для ключів і **N + 1** місце для вказівників плюс лічильник, а також тут не виділяється місце для вказівника на наступну сторінку. Крім того, ми виділяємо ще одне місце для ключа, а інше для значення для забезпечення можливості вставки. Підсумовуючи, для кожної сторінки потрібна однакова кількість вільних позицій: **2 * (N + 2)**. Всі B +-дерева можуть бути виділені як двомірний масив або як сукупність цих масивів, бронюючи собі GPU пам’ять. Кожна сторінка займає два рядки, тоді як елементи у сторінках формують колонки масиву. Це показано на рис. 1.3: темно-сірими клітинками позначено звичайні невикористані місця, призначені для вставки; світло-сірі клітини позначають значення, котрі зберігаються в листах дерева.

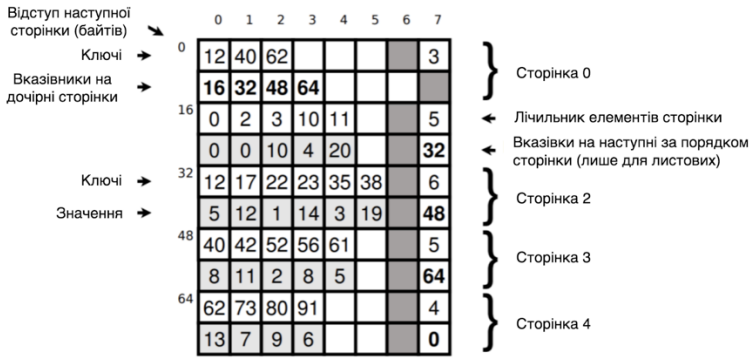

Рисунок 1.3 – схематична структура інвертованого індексу [8]

Кожнен лист сторінки має одну клітинку з вказівником на наступну сторінка, відповідно, на останній сторінці аркуша цей вказівник дорівнює NULL. Властивості багатоядерних GPU гарантують максимальну продуктивність, якщо система може тримати всі ядра зайняті задачами протягом всього часу обробки. Якщо віддати велику кількість блоків потоків одному потоковому процесору, компілятор і апаратне забезпечення організує обробку даних автоматично таким чином, що якщо один блок потоків чекає на дані, інші можуть продовжувати свою роботу. Затримка пам'яті тоді прихована і, відповідно, таким чином можна отримувати значне

прискорення у роботі індексу. Недоліком цього рішення є те, що порядок виконання блоків потоків невідомий, і це вимагає брати до уваги додаткові обмеження при реалізації нетривіальних алгоритмів. Відповідно, досягнення максимальної пропускної спроможності інструкцій може бути неможливою через складну синхронізацію між потоками, що працюють з різними блоками. Тому обробка B+-дерева бути паралельною на одній сторінці, але не повинна працювати паралельно між кількома сторінками одночасно, оскільки, наприклад, вставка змінює структуру і можеконфліктувати з іншим потоком, що виконують вставку на інші (чи ту ж) сторінку. Як бачимо, для цього необхідний складний механізм блокування сторінок, що може буде не ефективним для роботи GPU. Також, GPU може швидше отримувати доступ до пам'яті, якщо всі потоки в блоці читають або записують байти в межах одного сегменту. У таких випадках можна об'єднати доступ до пам'яті. Якщо дозволити кожному потоку читати з віддаленого місця в пам'яті (наприклад, під час алгоритму паралельного пошуку), всі доступи до пам’яті будуть послідовно знижувати пропускну спроможність пам’ятіи. Отже, для максимально ефективної паралелізації обробки за допомогою потоків GPU система має працювати в межах однієї сторінки, а навігація та управління обробкою дерева в цілому повинно бути на боці зоста. Очевидно також, що якщо ми хочемо використати всю потужність потокових процесорів, кількість елементів на одній сторінці повинна бути достатньо великою, щоб тримати всі ядра GPU зайнятими під час обробки однієї сторінки. З іншого боку, скорочення розміру сторінки прискорює пошук і вставку елементів у дерево. [8]

1.7 Висновок

Таким чином, основною класичною індексною структурою для діапазонного пошуку є B-tree. Для цієї задачі можна використовувати і інші індексні структури, але B-tree зазвичай є найбільш ефективною.

Основною проблемою роботи B-tree є те, що зі збільшенням кількості елементів у наборі даних збільшується і глибина дерева, а разом з ним логарифмічно зростає і

час пошуку по індексу. Вдаючись до використання GPU для роботи з даними можна значно пришвидшити час пошуку застосовуючи паралелізацію, але все одно у неї існують фізичні обмеження, адже неможливо створити графічний процесор з кількістю ядер, що буде дорівнювати кількості елементів набору даних. Тому очевидно, що схема пошуку даних має бути переосмислена, наприклад, застосовуючи методи машинного навчання для побудови предиктивних моделей пошуку позиції елемента у індексі.

2 ВИКОРИСТАННЯ ЗАСОБІВ МАШИННОГО НАВЧАННЯ ДЛЯ МОДЕЛЮВАННЯ САМОНАЛАГОДЖУВАЛЬНИХ ІНДЕКСНИХ СТРУКТУР

2.1 B-tree як предиктивна модель

Найпоширенішою структурою індексу для визначення розташування ключа у сортованому масиві є B-Tree. B-Tree - збалансовані та кеш-ефективні дерева. Вони відрізняються від більш поширених дерев, таких як бінарні дерева, в тому, що у кожного вузла є досить великий розгалужувальний коефіцієнт (наприклад, 100), щоб відповідати розміру сторінки для ефективної ефективної ієрархії пам'яті (тобто вертикальні вузли дерева завжди залишаються в кеші). Таким чином, для кожного оброблюваного вузла модель отримує точний приріст у 100. Звичайно, обробка вузла вимагає часу. Як правило, обхід одного вузла розміру 100, припускаючи, що він у знаходиться у кеш пам’яті, займає приблизно 50 циклів для сканування сторінки (припускаючи, що це сканування, так як це зазвичай швидше, ніж бінарний пошук в таких розмірах).

По суті, до B-Trees відносяться моделі виду *Ф (ключ) → позиція*. У даній роботі використано більш загальну нотацію машинного навчання, де ключ буде представлений як **x**, а позиція – як **y**. Оскільки ми знаємо всі дані в базі на час навчання (час побудови індексу), нашою метою є вивчення моделі f (x) ≈ y. Цікаво, що оскільки дані сортуються, f моделює кумулятивну функцію розподілу (Cumulative Distribution Function - CDF) даних. Проблема моделювання розподілу такими функціями давно досліджується. [2]

2.2 Машинне навчання

Машинне навчання - це область комп’ютерних наук, що використовує методи та підходи теорії ймовірності і статистики для того, щоб дати техніці можливість

“навчатись”, тобто послідовно покращувати продуктивність роботи над тими чи іншими задачами, за допомогою даних, не будучи прямо запрограмованим відповідним алгоритмом. Область машинного навчання виникла з робіт із розпізнавання шаблонів та обчислювальної теорії навчання у галузях штучного інтелекту, і займається дослідженням та створенням алгоритмів, що можуть вчитись на певних даних та робити на них передбачення - такі алгоритми працюють краще, ніж класичні статичні програми використовуючи передбачення або приймаючи рішення базуючись на реальних даних.

Машинне навчання знайшло широке застосування у ряді обчислювальних задач, де розробка та програмування явних статичних алгоритмів, що працювали б достатньо добре, є задачою складною або невирішуваною. Прикладом таких задач може бути фільтрація емейлів, розпізнавання облич, розпізнавання симолів, навчання ранжуванню.

Машинне навчання тісно пов’язане з обчислювальною статистикою, яка також фокусується на передбаченню результатів за допомогою комп’ютерів, та математичними методами оптимізації, які надають теорію та алгоритми для оптимізації цільових функцій навчання.

Загалом, машинне навчання можна поділити на кілька основних підходів:

- навчання з учителем
- навчання без учителя

Розглянемо їх детальніше.

2.2.1 Навчання з учителем

Основною особливістю алгоритмів навчання з учителем є те, що дані, які подаються на вхід, розмічені певним чином. Загалом, за допомогою навчання з учителем можна вирішувати задачі регресії та класифікації.

Почнемо із задачі ***класифікації***. Метою класифікації є вивчення відображення входів $\boldsymbol{x}$ у виходи $\boldsymbol{y}$, де $\boldsymbol{y} \in \{\boldsymbol{1}, \ldots, \boldsymbol{C}\}$, а C - кількість класів. Коли $C = 2$, маємо задачу бінарної класифікацієї (в цьому випадку найчастіше приймають, що $y \in \{0, 1\}$); якщо $C > 2$ отримуємо задачу мультикласової класифікації.

Одним із способів формалізувати проблему є апроксимація функції. Припустимо, що $y = f(x)$ - деяка невідома функція f, а мета навчання полягає в оцінці функції f за умови наданого розміченого тренувального набору даних. Базуючись на отриманій оцінці, ми отримуємо $\hat{y} = \hat{f}(x)$ (де ^ - символ оцінки). Головна ціль - робити правдиві прогнози щодо нових вхідних даних, тобто тих, яких алгоритм пошуку оцінки не бачив під час тренування (загалом, ми намагаємось *узагальнити* модель), оскільки прогнозувати відповідь на вхідні дані з тренувального набору легко (адже алгоритм може просто знати відповідь). [9]

Зазвичай, навчальний алгоритм замість чіткого результату повертає ймовірність, яка відповідає кожному класу для заданих вхідних даних. Позначимо розподіл ймовірностей над відомими мітками за заданим вхідним вектором $\boldsymbol{x}$ та набором тренувальних даних $\boldsymbol{D}$ як $p(y|x, D)$. Загалом, цей розподіл представляється векторами розміру C. Однак, у випадку двох класів достатньо повертати лише одне значення $p(y = 1\,|x, D)$, адже $p(y = 1\,|x, D) + p(y = 0\,|x, D) = 1$. У даній нотації чітко зазначено, що ймовірність умовно залежить від вхідних даних $\boldsymbol{x}$, а також від тренувального набору $\boldsymbol{D}$. Також ймовірність неявно залежить від форми моделі, яка використовується для прогнозування. Тобто, для вибору між різними моделями робиться припущення шляхом явного написання $p(y|x, D, M)$, де M позначає модель. [9]

З огляду на ймовірнісний вихід, можна обчислити "найкращу здогадку" як "справжню мітку", використовуючи

$$\hat{y} = \hat{f}(x) = argmax_{c=1}^{C} p(y = c|x, D) \qquad (1)$$

Це відповідає найбільш вірогідній класовій мітці і називається режимом розподілу $p\ (y \mid x, D)$; він також відомий як *оцінка MAP* (де *MAP* - максимальну апостеріорну оцінку).

Класифікація є, мабуть, найбільш широко використовуваною формою машинного навчання, і була використана для вирішення багатьох цікавих і часто важких реальних проблем, до прикладу, розпізнавання символів, розпізнавання

облич, аналіз тональності документів або фільтрація спаму серед електронних листів.

Задача **регресії** фактично нічим не відрізняється від задачі класифікації за виключенням того, що вихідна змінна є нескінченою, а не дискретною. На рис. 2.1 можемо бачити простий приклад: один дійсний вхід $x_i \in R$ та один дійсний вихід $y_i \in R$. В даному випадку розглядається дві моделі для оцінки даних: пряма лінія та квадратична функція. До цієї задачі можна додати багато інших умов, таких як високорозмірні дані, викиди, негладкі вихідні сигнали і т. д.

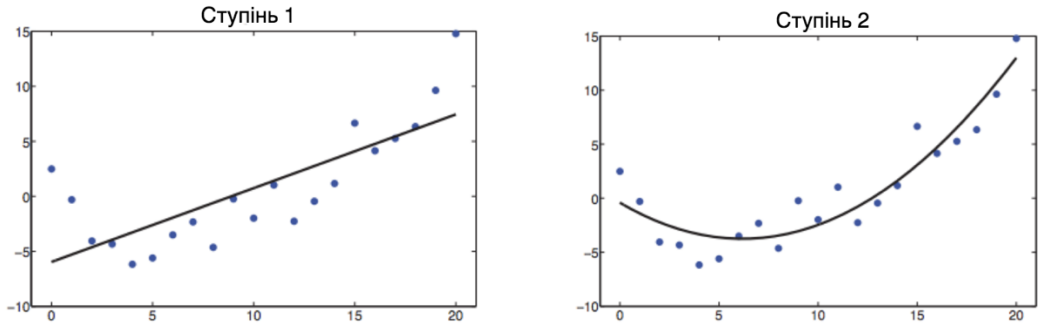

Рисунок 2.1 — Апроксимація даних за допомогою лінійної регресії поліномами першого та другого ступеню [9]

Прикладами задач, де можна використати регресійні методи навчання, є:

- передбачення коливання ринкових цін
- передбачення віку людей, що переглядають відео
- передбачення положення у просторі руки робота за допомогою сигналів з різних датчиків
- передбачення температури

Підсумовуючи написане вище: навчання з учителем намагається вивчити відображення з простору вхідних даних у простір вихідних на основі певного набору розмічених даних та намагається узагальнити це відображення.

2.2.2 Навчання без учителя

Основною ідеєю навчання без учителя є те, що на вхід ці алгоритми беруть лише вхідні дані без будь-якої розмітки, на відміну від навчання з учителем. Основною метою є виявлення "цікавої структури" у даних; це іноді називають

відкриттям знань. Замість цього задача формалізується як побудова оцінки щільності розподілу вхідних, тобто побудова моделі форми $p(x_i|\theta)$.

Існує дві відмінності у порівнянні з навчанням з учителем. По-перше, нотація $p(x_i|\theta)$ замість $p(y_i|x_i,\theta)$; тобто навчання з учителем - це *умовна* оцінка щільності, тоді як навчання без учителя - *безумовна* оцінка щільності. По-друге, x_i - це *вектор* ознак, відповідно алгоритм має створити мультиваріантну ймовірнісну модель. Це значно відрізняється від навчання з учителем, адже там y_i - це, як правило, лише одна змінна, яку ми намагаємося передбачити. Отже, для більшості проблем, що вирішуються за допомогою навчання з учителем, ми можемо використовувати одновимірні моделі розподілу ймовірності, що значно спрощує проблему.

Навчання без учителя, безсумнівно, більш характерне для моделі навчання людей та тварин. Воно також застосовується більш широко, ніж навчання з учителем, оскільки воно не вимагає від людини-експерта ручної розмітки даних. Розмічені дані є не тільки дорогими для придбання, але також містять порівняно мало інформації, зазвичай, недостатньо для того, щоб надійно оцінити параметри комплексу моделі.

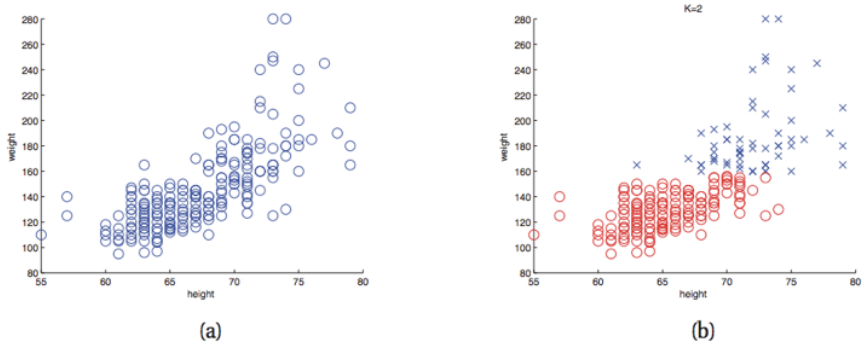

Рисунок 2.2 — а) візуалізація залежності ваги та росту людини; б) можливий варіант кластеризації на 2 кластера. [9]

Як канонічний приклад навчання без учителя, розглянемо проблему кластеризації даних у групи. Наприклад, на рис 2.2 (а) містяться деякі двухвимірні дані, що представляють висоту та вагу групи людей. Схоже, що можуть бути різні кластери або підгрупи, хоча незрозуміло, скільки. Нехай K позначає кількість кластерів. Наша перша мета - оцінити розподіл за кількістю кластерів, $p(K|D)$. Це

говорить нам, чи існують субпопуляції в даних. Для простоти ми часто наближаємо розподіл $p(K|D)$ за його *модою*, $K^* = argmax_K\ p(K|D)$. У випадку навчання з учителем ми знаємо, що існує два класи (чоловік і жінка), але у випадку навчання без учителя ми можемо вибирати будь-яку кількість груп. Відбір моделі "правильної" складності називається *вибором моделі*.

Друга мета - визначити, до якої групи належить кожна точка. Нехай $z_i \in \{1, \ldots, K\}$ - це кластер, якому належить i точка серед даних. (z_i є прикладом прихованої змінної, тому що вона не спостерігається в навчальному наборі). Ми можемо визначити, якому кластеру належить дана точка, обчислюючи

$$z_i^* = argmax_k\ p(z_i = k|x_i, D) \qquad (2)$$

Це проілюстровано на рис. 2.2 (б), де використані різні кольори для позначення класів, припускаючи, що $K = 2$.

2.3 Алгоритми машинного навчання

2.3.1 Штучні нейронні мережі

Штучна нейронна мережа (ANN) - це парадигма обробки інформації, яка надихається методом сприймання даних у біологічної нервової системи, таких як мозок. Ключовим елементом цієї парадигми є нова структура системи обробки інформації. Вона складається з великої кількості сильно взаємопов'язаних процесорних елементів (нейронів), які працюють разом для вирішення конкретних проблем. Нейронні мережі, як і люди, навчаються на прикладах. Нейромережі налаштовуються для певної задачі, наприклад, розпізнавання образів або класифікація даних, через процес навчання. Навчання в біологічних системах передбачає коригування синаптичних зв'язків, які існують між нейронами - схожим чином працюють також і штучні нейронні мережі.

Складність реальних нейронів дуже абстрагована при моделюванні штучних нейронів. Вони в основному складаються з входів (наприклад, синапсів), які множаться на ваги (сила відповідних сигналів), а потім передаю ться у математичну

функцію, яка визначає активацію нейрона. Нейронні мережі об’єднують багато штучних нейронів для обробки інформації.

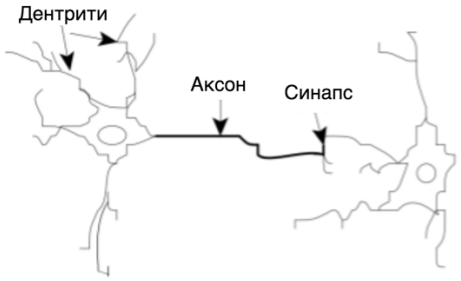

Рис. 2.3 — Графічна структура біологічного нейрону [10]

Чим більша вага штучного нейрона, тим більше буде підсилюватись вхід, який перемножиться на нього. Ваги також можуть бути негативними, тому можна сказати, що сигнал гальмується негативною вагою. Залежно від ваг, обчислення нейрона буде різним. Регулюючи ваги штучного нейрона, ми можемо отримати вихід, який потрібно отримати для конкретних входів. Але коли у нас є штучна нейронна мережа з сотнями або тисячами нейронів, досить складно знайти вручну всі необхідні ваги. Для цього потрібні алгоритми, які можуть регулювати ваги мережі для отримання бажаного виходу. Цей процес налагодження ваг називається *навчанням*. Існує велика кількість нейронних мереж. З часу виходу першої нейронної моделі МакКуллоха та Пітса (1943) було розроблено сотні різних моделей, що розглядаються як нейронні мережі. Вони відрізняються функціями активації, прийнятими значеннями, топологією, алгоритмами навчання тощо. Також є багато гібридних моделей, де кожен нейрон має більше властивостей, ніж ті, що ми розглядаємо тут. [10]

Детальніше розглянемо внутрішню структуру нейрону.

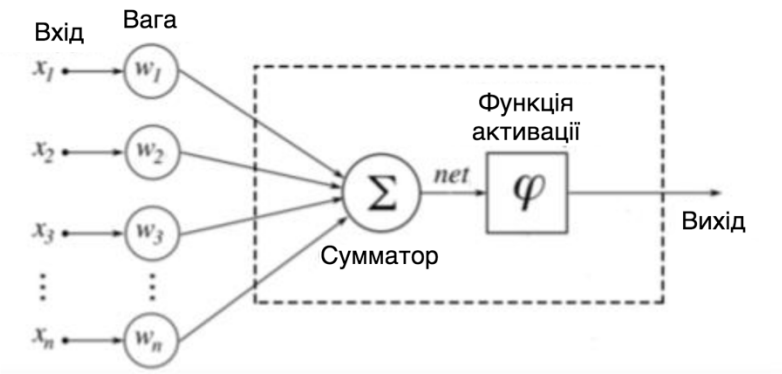

Рис. 2.4 — Графічна структура штучного нейрону [11]

У кожного нейрона, в тому числі і у штучного, повинні бути якісь входи, через які він приймає сигнал. Вище було згадано ваги, на які множаться сигнали, що проходять по зв'язку. На рисунку вище ваги зображені кружками.

Сигнали, що надійшли на вхід, множаться на свої ваги. Сигнал першого входу x_1 множиться на відповідну вага w_1. У підсумку отримуємо $x_1 w_1$. І так до n-ого входу. У підсумку на останній вхід отримуємо $x_n w_n$.

Після цього всі множники передаються в акумулятор. Уже виходячи з його назви можна зрозуміти, що він робить. Він просто підсумовує все вхідні сигнали, помножені на відповідні ваги:

$$x_1 w_1 + x_2 w_2 + \cdots + x_n w_n = \sum_{i=1}^{n} x_i w_i \qquad (3)$$

Результатом роботи суматора є число, яке називають *зваженою сумою*. Роль суматора очевидна - він агрегує всі вхідні сигнали (яких може бути багато) в якесь одне число - зважену суму, яка характеризує, чи надійшов на нейрон сигнал в цілому. Ще зважену суму можна уявити як ступінь загального збудження нейрона.

Просто так подавати виважену суму на вихід досить безглуздо. Нейрон повинен якось обробити її і сформувати адекватний вихідний сигнал. Саме для цих цілей і використовують функцію активації. Вона перетворює зважену суму в якесь число, яке і є виходом нейрона (вихід нейрона позначимо за допомогою змінної out).

Для різних типів штучних нейронів використовують найрізноманітніші функції активації. У загальному випадку їх позначають символом φ (net). Вказівка

зваженого сигналу в дужках означає, що функція активації приймає зважену суму як параметр.

Отже, *функція активації* (Activation function, $\varphi(net)$) - функція, що приймає зважену суму як аргумент. Значення цієї функції і є виходом нейрона (out).

$$out = \varphi(\text{net}) \quad (4)$$

Розглянемо найпопулярніші функції активації.

Функція одиничного стрибка

Найпростіший вид функції активації, зображена на рис. 2.5. Вихід нейрона може дорівнювати лише 0 або 1. Якщо зважена сума більше певного порогу b, то вихід нейрона дорівнює 1. Якщо нижче, то 0.

$$out(net) = \begin{cases} 0, net < b \\ 1, net \geq b \end{cases} \quad (5)$$

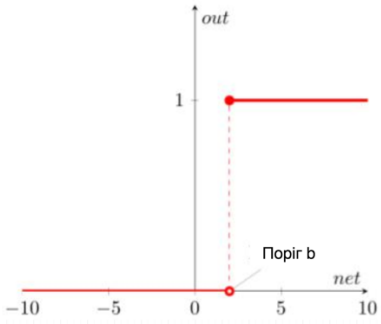

Рис. 2.5 — Активаційна функція одиничного стрибка

На горизонтальній вісі розташовані величини зваженої суми. На вертикальній осі - значення вихідного сигналу. Як легко бачити, можливі тільки два значення вихідного сигналу: 0 або 1. Причому 0 буде видаватися завжди від мінус нескінченності і аж до деякого значення зваженої суми, званого порогом. Якщо зважена сума дорівнює порогу або більше нього, то функція видає 1.

Сигмоїдальна функція

Насправді існує ціле сімейство сигмоїдальних функцій, деякі з яких застосовують як функції активації в штучних нейронах, зображена на рис. 2.6.

Всі ці функції мають деякі дуже корисними властивостями, заради яких їх і застосовують в нейронних мережах.

$$out(net) = \frac{1}{1 + \exp(-a * net)} \quad (6)$$

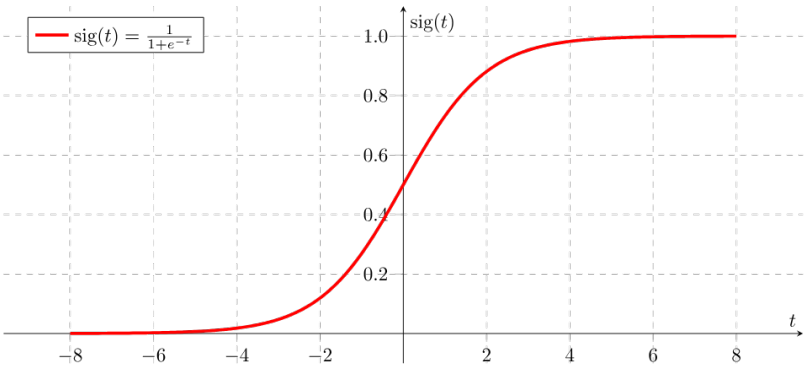

Рис. 2.5 — Активаційна функція сигмоїд

Функції цього сімейства мають кілька важливих властивостей:

- вона є «стискає» функцією, тобто незалежно від аргументу (зваженої суми), вихідний сигнал завжди буде в межах від 0 до 1
- вона більш гнучка, ніж функція одиничного стрибка - її результатом може бути не тільки 0 і 1, але і будь-яке число між ними
- у всіх точках вона має похідну, і ця похідна може бути виражена через цю ж функцію

Саме через ці властивості логістична функція найчастіше використовуються в якості функції активації в штучних нейронах.

Випрямлена лінійна одиниця - ReLU

У контексті штучних нейронних мереж лінійна виправлена одиниця (рис. 2.6) - це функція активації, визначена як позитивна частина її аргументу:

$$f(x) = x^+ = \max(0, x) \quad (7)$$

де x - вхід нейрона. Ця функція активації була вперше представлена в динамічній мережі Hahnloser сильними біологічними мотивами та математичними обґрунтуваннями. Вперше в 2011 році було продемонстровано можливість кращої підготовки глибших мереж [12] у порівнянні з широко використовуваними функціями активації до 2011 року, тобто логістичною сигмоїдою і його більш практичний [13] аналог, гіперболічна дотична точка. Лінійна виправлена одиниця

станом на 2018 р. є найпопулярнішою функцією активації для глибоких нейронних мереж.

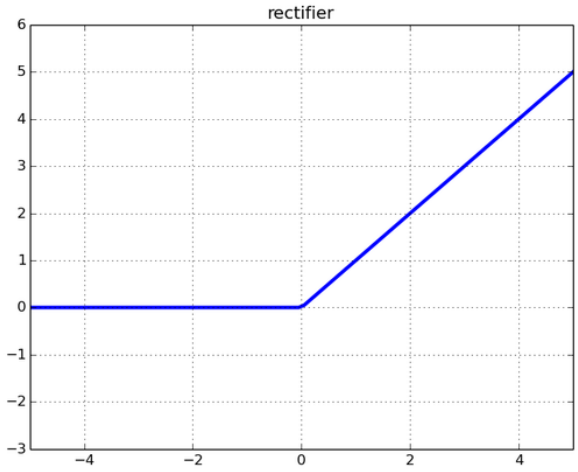

Рис. 2.6 — Активаційна функція ReLU

Структура, що використовує випрямляч, також називається випрямленою лінійною одиницею (ReLU). Існує кілька підвидів ReLU, такі як зашумлений ReLU, нещільний ReLU, та ELU - експоненційна лінійна структура.

Також, у 2017 році група вчених розробила ще одну активаційну функцію, яка на здивування гарно працює з повнозв’язними нейронннми мережами - SELU - scaled exponential linear units (рис. 2.7).

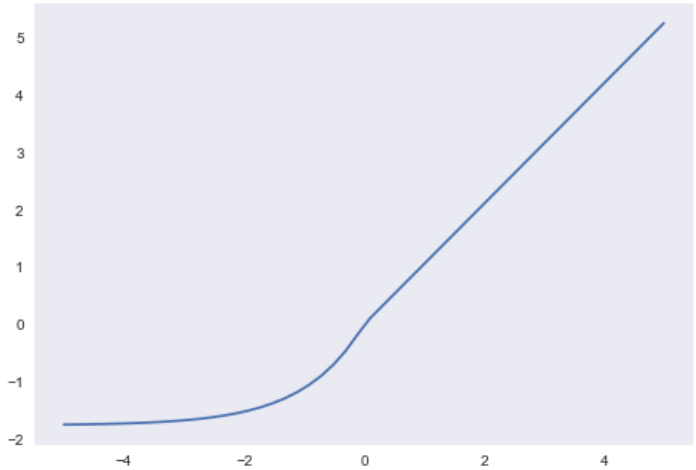

Рис. 2.7 — Активаційна функція SELU

Виявляється, для нормалізованих вхідних даних експериментально доведено, що значення lambda та alpha мають бути **1.67326324235437728481704299167 17** та **1.05070098735548049341933498529 46** відповідно.

$$selu(x) = \lambda \begin{cases} x & if\ x > 0 \\ \alpha e^x - \alpha & if\ x \leq 0 \end{cases} \quad (8)$$

У даній роботі буде використано повнозв’язні нейронні мережі, адже на вхід моделі нам подається маловимірні вхідні дані. Розглянемо детальніше їх структуру.

Як правило, в більшості нейронних мереж є так званий вхідний шар, який виконує тільки одну задачу - розподіл вхідних сигналів іншим нейронам. Нейрони цього шару не проводять ніяких обчислень.

В одношарових нейронних мережах сигнали з вхідного шару відразу подаються на вихідний шар. Він проводить необхідні обчислення, результати яких відразу подаються на виходи.

Виглядає одношарова нейронна мережа наступним чином (рис. 2.8):

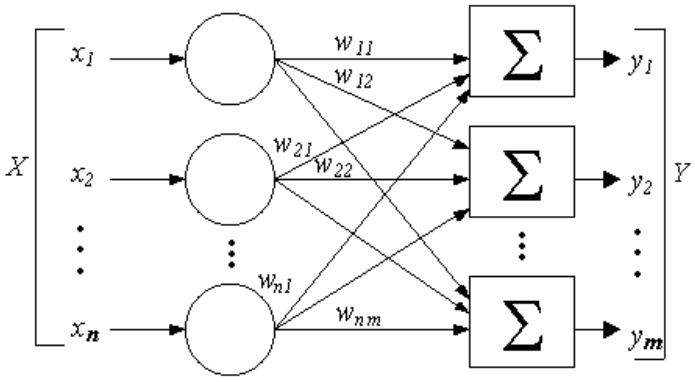

Рис. 2.8 — Одношарова повнозв’язна нейронна мережа. [11]

На цій картинці вхідний шар позначений кружками (він не вважається за шар нейронної мережі), а праворуч розташований шар звичайних нейронів.

Нейрони з'єднані один з одним стрілками. Над стрілками розташовані ваги відповідних зв'язків (вагові коефіцієнти).

Багатошарові нейронні мережі, крім вхідного і вихідного шарів нейронів, характеризуються ще і прихованим шаром (шарами). Зрозуміти їх розташування просто - ці шари знаходяться між вхідним і вихідним шарами.

Така структура нейронних мереж копіює багатошарову структуру певних відділів мозку.

Назва прихований шар отримав невипадково. Справа в тому, що тільки відносно недавно були розроблені методи навчання нейронів прихованого шару. До цього обходилися тільки одношаровими нейросетями.

Багатошарові нейронні мережі мають набагато більші можливості, ніж одношарові.

Вихідний сигнал збирається за стадіями. Після кожного шару виходить якийсь проміжний результат. Приховані шари теж перетворять вхідні сигнали в деякі проміжні результати. [11]

2.3.2 Метод опорних векторів

Метод опорних векторів (SVM – support vector machine) - це сімейство алгоритмів машинного навчання, які використовуються для задач класифікації та регресії. Це популярний інструмент для класифікації та регресії, який вперше був визначений Володимиром Вапніком та його колегами у 1992 році [14]. Регресія SVM розглядається як непараметрична техніка, оскільки вона спирається на функції ядра.

Припустимо, що у нас є набір тренувальних даних, де x_i являє собою багатоваріантний набір N спостережень із спостережуваними значеннями відповіді y_n.

Спочатку потрібно знайти лінійну функцію

$$f(x) = x'\beta + b, \quad (9)$$

і переконатись, що вона максимально рівна. Далі потріюно знайти $f(x)$ з мінімальною величиною норми $(\beta'\beta)$. Це сформульовано як завдання опуклої оптимізації для мінімізації:

$$J(\beta) = 12\beta'\beta \quad (10)$$

що підлягає всім залишках, що мають значення менше ε; або, у формі рівняння:

$$\forall n: yn - (xn'\beta + b) \leq \varepsilon. \quad (11)$$

Можливо, що такої функції f (x) немає для задоволення цих обмежень для всіх точок. Для вирішення інших недосяжних обмежень введіть слабкі змінні ξ_n і ξ^* n для кожної точки. Цей підхід схожий на концепцію "м'якого відступу" в класифікації SVM, тому що змінні величини логіки дозволяють помилкам регресії існувати включно до значеннь ξ_n і ξ^*, але все-таки задовольняють необхідним умовам. [15]

Метод опорних векторів також може бути використана як метод регресії, зберігаючи всі основні функції, що характеризують алгоритм (максимальний запас). Метод регресії опорних векторів (SVR) використовує ті самі принципи, що і SVM для класифікації, з лише кількома незначними відмінностями. Перш за все, оскільки вихід є дійсним числом, дуже важко прогнозувати наявну інформацію, яка має нескінченні значення. У випадку регресії епсилон – порогове значення встановлюється в наближенні до SVM. Але, крім цього факту, є також і більш складна причина, тому алгоритм більш складний, тому слід враховувати. Однак основна ідея завжди однакова: мінімізувати помилку, індивідуалізувати гіперплощу, що максимізує маржу, маючи на увазі, що частина помилки переноситься (рис. 2.9).

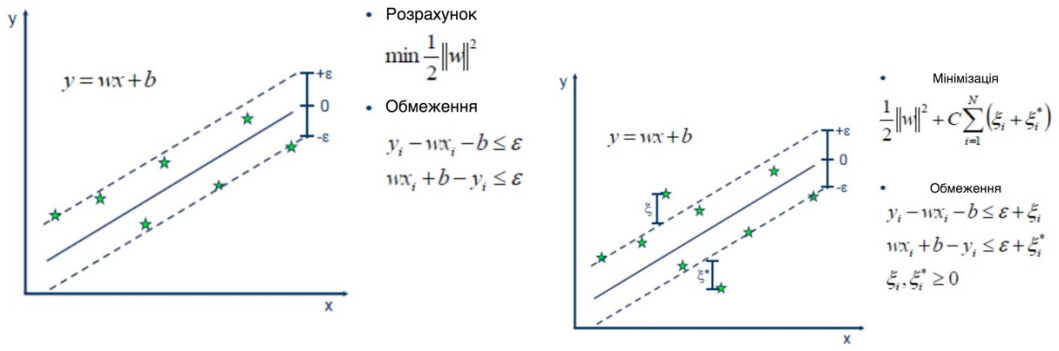

Рис. 2.9 — Принцип роботи методу опорних векторів [15]

2.4 Вибір архітектури моделі

Як у розв’язанні регресійної проблеми, ми навчаємо нашу модель $f(x)$ з квадратичною помилкою, щоб мати змогу передбачити позицію y. Питання в тому,

яку архітектура моделі використовувати для f. Як згадано вище, для того, щоб модель була успішною, вона повинна підвищити точність прогнозування y у коефіцієнті більше ніж 100 за менш ніж 50 циклів процесора. З багатьох мільйонів прикладів, що потребують високої точності, побудова однієї широкої та глибокої нейронної мережі часто не дає високої точності, та і зазвичай вона обчислювально дорога у використанні. Взамін, запропонована побудова ієрархії моделей, натхненна роботою низки експертів. Для початку припустимо, що $y \in [0, N]$ і на кроці l ми маємо M_l моделей. Ми тренуємо модель на кроці 0, $f_0(x) = y$. Далі, модель k на кроці l, позначена як $f_l^{(k)}$, тренується з цільовою функцією

$$L_l = \sum_{(x,y)} \left(f_l^{(M_l * f_{l-1}(x)/N)} - y \right)^2, L_0 = \sum_{(x,y)} (f_0 - y)^2, \tag{12}$$

Зауважимо, що тут використовується позначення $f_{l-1}(x)$, рекурсивно виконуючи $f_{l-1}(x) = f_{l-1}^{\left(\left\lfloor M_{l-1} * \frac{f_{l-2}(x)}{N} \right\rfloor\right)}(x)$. Таким чином, загалом, ми повторно тренуємо кожен етап з функцією втрат L_l для побудови повної моделі. Варто зазначити, що моделі потрібно тренувати ітеративно, оскільки сучасним фреймворкам машинного навчання складно мастшабуватись до графу обчислень з кількома тисячами вузлів (рис. 2.10).

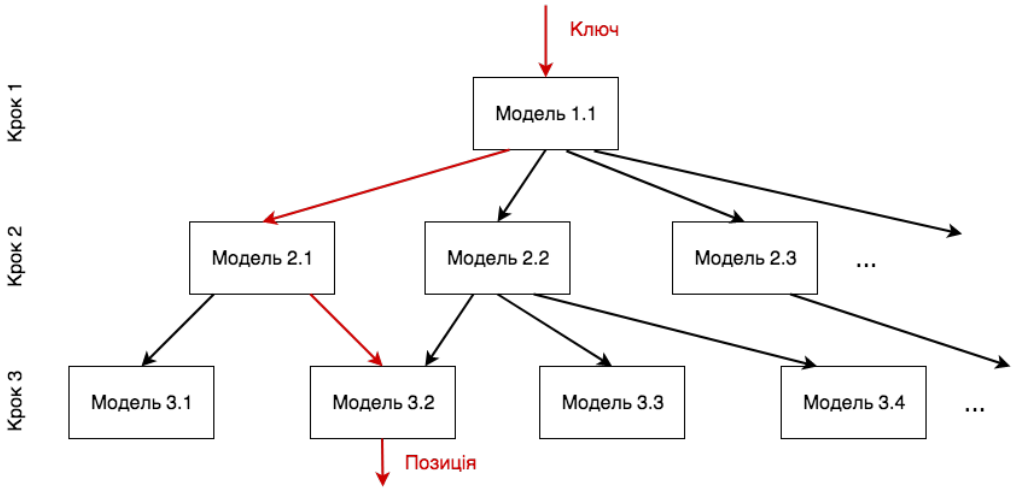

Рис. 2.10 — Приклад архітектури моделі навченого індексу

2.5 Потенційні обмеження

На відміну від типових ML-моделей, недостатньо добре знайти приблизне рішення. Швидше за все, у момет статистичного виведення, коли модель передбачає оцінку позиції, ми повинні знайти фактичний запис, який відповідає ключеві запиту. Традиційно B-Tree забезпечують розташування початку сторінки, на якій знаходиться ключ. Для навчених індексів ми повинні шукати окіл передбачення, щоб фактично знайти початок діапазону.

Цікаво, що ми можемо легко обмежити помилку нашої моделі так, щоб можна було використовувати класичні алгоритми пошуку, такі як бінарний пошук. Це можливо завдяки тому, що ми знаємо під час тренування всі ключі, які індексує модель. Таким чином, ми можемо обчислити максимальну похибку Δ, яку модель робить у сумі над усіма ключами, і на час виведення виконаємо пошук в діапазоні $[y - \Delta, y + \Delta]$. Далі, ми можемо розглянути помилку моделі над підмножинами даних. Для кожної моделі k на останньому етапі нашої загальної моделі ми обчислюємо її помилку Δk. Це виявиться досить корисним, оскільки деякі частини моделі є більш точними, ніж інші, і, коли Δ зменшується, час пошуку зменшується. Цікавим майбутнім напрямком є мінімізація найгіршої помилки Δ, а не середньої помилка. У деяких випадках також можна побудувати менші B-дерева, щоб піти від кінця прогнозування моделі до остаточної позиції. Ці гібридні моделі часто менш ефективні у плані пам'яті, але можуть підвищити швидкість завдяки ефективності кеш-пам'яті. [2]

2.6 Використання методів Байєсівської оптимізації

Маючи у розпорядженні загальний підхід для побудови моделі, потрібно розуміти складність вибору найкращої моделі серед можливих варіантів гіперпараметрів: кількості шарів, розмір сторінки, тип алгоритму передбачення на “останній милі”. Загалом, цей список можна сильно розширити.

Звичайно, можна скористатись *перебором сітки* гіперпараметрів, але цей метод може бути надто затратним у використанні в залежності від кількості гіперпараметрів, адже складність перебору буде рости квадратично.

Для вирішення цієї задачі можна скористатись менш енергозатратним та ефективнішим способом за допомогою Гаусівських процесів та методів баєсівської оптимізації.

Як і в інших алгоритмах оптимізації, під час Байєсівської оптимізації ми зацікавлені у тому, щоб знайти мінімум функції $f(x)$ на обмеженій множині параметрів X. Основна відмінність цього алгоритму полягає у тому, що ми будуємо ймовірнісну модель для функції $f(x)$, а потім використовуємо цю модель для визначення, в якому наступному місці множини X оцінювати цільову фукнцію. Важливим моментом при використанні цього алгоритму є також те, що потрібно враховувати всю попередньо відому інформацію про цільову функцію, а не тільки локальний градієнт чи гессіан. В результаті цього, можна знаходити мінімуми доволі складних неопуклих функцій за відносно невелику кількість обчислень значень цих функцій. [16]

Цільову функцію для оптимізації гіперпараметрів моделей машинного навчання доволі складно обчислювати, адже це потребує повноцінного тренування такої моделі. Задача ще більш ускладнюється при тренуванні доволі складних моделей, якими зазвичай є нейронні мережі.

Враховуючи зазначену вище інформацію, можна зробити висновок, що для пошуку гіперпараметрів моделей машинного навчання, використання методу баєсівської оптимізації є доволі доцільним. Внаслідок цього, в цій роботі буде використовуватись саме цей алгоритм.

2.7 Висновки

Отже, для досягнення моделі пошуку за константний було запропоновано схему побудови самоналагоджувальної індексної структури, робота якої ґрунтується на застосуванні методів машинного навчання.

Основною ідеєю є використання функції розподілу даних у ролі предиктивної моделі позиції у вісортованому масиві даних. Для вивчення функції розподілу даних можна використовувати одну із моделей машинного навчання. При умові, що така модель достатньо точно вивчить функцію розподілу, кількість даних, яку індексу потрібну буде проглянути для пошуку, буде настільки малою, що асимптотичний час пошуку можна буде вважати константим.

Зважаючи на те, що предиктивна модель має працювати швидко та має займати у пам’яті місця не набагато більше, ніж звичайний B-tree індекс, будуть використані доволі прості моделі машинного навчання, такі як неглибокі повнозв’язні нейронні мережі та метод опорних векторів, які потім будуть поєднані у модель типу гурт експертів з використанням Баєсівської оптимізації гіперпараметрів.

3 РЕАЛІЗАЦІЯ АЛГОРИТМУ ДЛЯ САМОНАЛАГОДЖУВАЛЬНОЇ ДІАПАЗОННОЇ ІНДЕКСНОЇ СТРУКТУРИ ТА ОГЛЯД РЕЗУЛЬТАТІВ

3.1 Огляд програмних засобів і бібліотек для реалізації алгоритму

3.1.1 Jupyter Notebook

Існує велика кількість програмних засобів та бібліотек для проведення експериментів з машинного навчання. Розглянемо основні варіанти, які будуть використані у даній роботі, їх переваги та недоліки.

Розробка експериментів велась у інтерактивному середовищі для досліджень Jupyter.

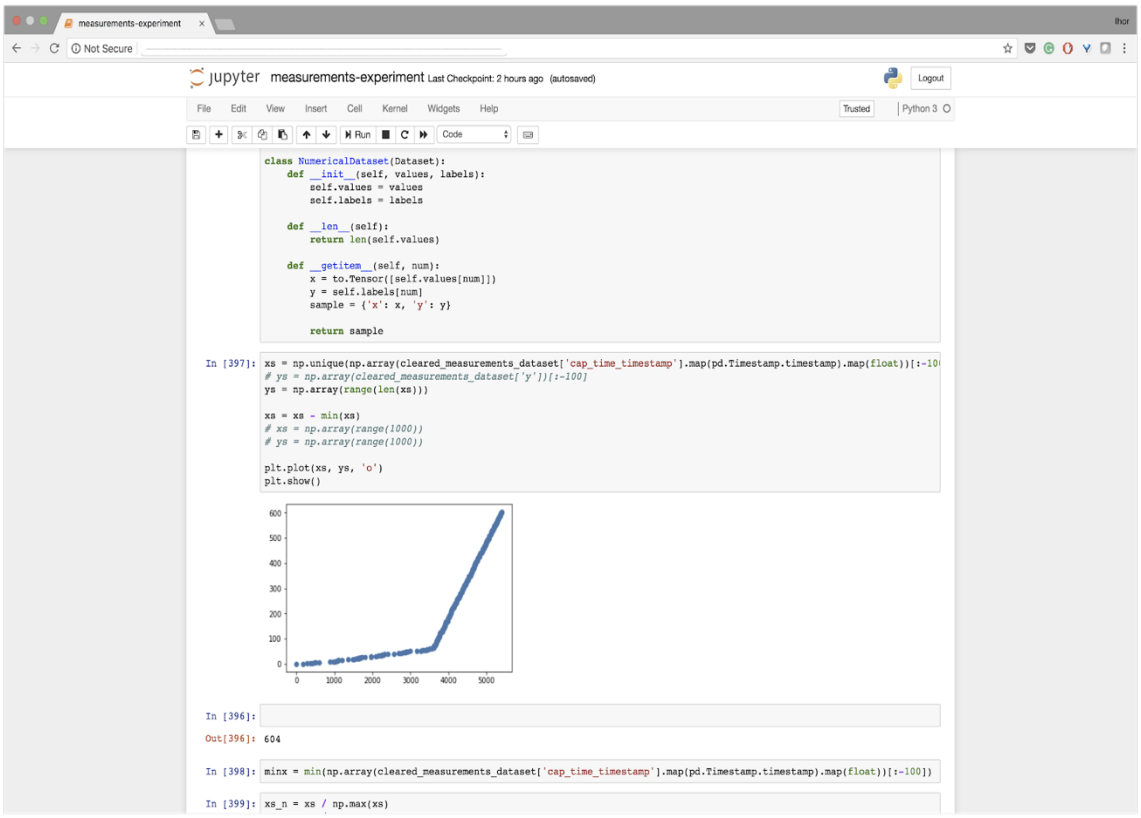

Рис. 3.1 — Дослідницьке середовище Jupyter

Ноутбук розширює підхід на основі консолі до інтерактивних обчислень у якісно новому напрямку за допомогою веб-додатку, придатного для побудови

всього обчислювального процесу: розробки, документування та виконання коду, а також передачт результатів. Jupyter Notebook поєднує в собі два компоненти:

Веб-застосунок: веб-інструмент для інтерактивного написання документів, що поєднують пояснювальний текст, математику, обчислення та їх мультимедійний випуск.

Документація: представлення всього вмісту, доступного у веб-додатку, включаючи входи та виходи експериментів, пояснювальний текст, формули, зображення та відображення мультимедійних матеріалів.

Jupyter працює на концепції ядер: для кожного ноутбука створюється своє ядро на потрібній розробнику мові. Експерименти у даній роботі проводились за допомогою мови програмування **Python.**

Jupyter Notebook на сьогоднішній день є де-факто стандартом у науковій розробці, і це повністю підтвердилось під час розробки експериментів у даній роботі. [17]

3.1.2 Фреймворки для побудови алгоритмів машинного навчання

Створення моделей машинного навчання з нуля є задачею дуже складною, адже у цьому випадку потрібно самостійно враховувати дуже багато низькорівневих нюансів алгоритмів, пов’язаних з матричними обчисленнями, методами оптимізації, роботи з GPU та ефективної утилізації можливостей апаратного програмного забезпечення. Тому для побудови моделей машинного навчання використовують високорівневі абстракції у вигляді фреймворків машинного навчання. На даний момент таких існує доволі багато, і у кожного є своє призначення, починаючи від засобу для швидкого прототипування Keras [18] та закінчуючи набором бібліотек та засобів для проведення повного циклу: від досліджень до використання на бойових проектах - Tensorflow від Google [19].

У даній роботі для побудови нейронних мереж та базових лінійних моделей був використаний популярний фреймворк для проведення досліджень PyTorch, розроблений у Facebook.

Рис. 3.2 — Логотип фреймворку для роботи з глибоким навчанням Pytorch

PyTorch - це Python пакет, який надає дві високорівневі функції:

- тензорні обчислення (схоже з Numpy [20]) з максимально ефективним GPU-прискоренням
- можливість створення глибоких нейронних мереж будь-якого типу, що базуються на системі автоматичного диференціювання

PyTorch надає унікальний інтерфейс для створення нейронних мереж. Більшість фреймворків, таких як TensorFlow, Theano, Caffe та CNTK, дають змогу будувати статичний граф виконання математичних операцій - відповідно, немає жодної змоги втрутитись у процес виконання програми після початку роботи, а для внесення змін потрібно зупиняти процес і запускати заново з новим кодом. Хоча цей код є декларативним і по ідеї має спрощувати процес, у випадку з побудовою таких величезних конструкцій, як нейронні мережі, ситуація навпаки ускладнюється. [21]

У PyTorch використовується техніка, яка називається автоматичною диференціацією у зворотньому режимі, яка дозволяє змінювати роботу вашої моделі на льоту. Хоча ця техніка не є унікальною для PyTorch, це одна з найшвидших її реалізацій на сьогодні. Користувач PyTorch отримує найвищу швидкість і гнучкість для своїх досліджень будь-якого рівня. [21]

У склад PyTorch входять наступні модулі:

- Torch – бібліотека для роботи з тензорами з шикорою підтримкою GPU
- Torch.autograd – бібліотека автоматичного диференціювання, що підтримує всі диференційовані тензорні операції, доступні у PyTorch
- Torch.nn – бібліотека для побудови глибоких нейронних мереж з використанням автоматичного диференціювання

- Torch.optim – пакет зі стандартними методами оптимізації, що базовані на використанні градієнтів (SGD, RMSProp, Adam, тощо)
- Torch.multiprocessing – надлаштування над стандартною бібліотекою для млуьтипроцесингу у Python з переробленою моделлю використання спільної пам'яті.

```
Import torch
N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
model = torch.nn.Sequential(
    torch.nn.Linear(D_in, H),
    torch.nn.ReLU(),
    torch.nn.Linear(H, D_out),
)
loss_fn = torch.nn.MSELoss(size_average=False)
learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(), lr=learn-
ing_rate)
for t in range(500):
    y_pred = model(x)
    loss = loss_fn(y_pred, y)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Для побудови простіших лінійних чи базований на навчених ядрах моделей у даній роботі використовується визнана у дослідницькому світі бібліотека **Scikit-learn** [22].

Рис. 3.3 — Логотип фреймворку для роботи з алоритмами машинного навчання Scikit-learn

Вона побудована на основі бібліотеки швидких математичних обчислень у Python NumPy.

У Scikit-learn дуже простий інтерфейс, який відмінно підходить для проведення швидких експериментів, а також висока продуктивність, що дозволяє використовувати Scikit-learn на бойових проектах.

У даній роботі дана бібліотека використовується для експериментів з методом опорних векторів.

3.1.3 Робота з Баєсівською оптимізацією гіперпараметрів

Як було описано у попередньому розділі, Баєйсівська оптимізація може дозволити нам швидко та ефективно знаходити потрібну архітектуру та гіперпараметри моделі, специфічні для заданих наборів даних. У даній роботі було використано бібліотеку Bayesian Optimization [23]. Вона надає простий та зручний інтерфейс для оптимізації будь-якої заданої функції за заданим діапазоном відповідних параметрів.

fmfn/BayesianOptimization - це звичайний Python пакет, він дуже легкий у встановленні та конфігурації. Він такоє базується на згаданих вище бібліотеках NumPy та Scikit-learn.

```
from bayes_opt import BayesianOptimization
# знайдемо максимум простої квадратичної функції двох змінних
bo = BayesianOptimization(lambda x, y: -x ** 2 - (y - 1) ** 2 + 1,
{'x': (-4, 4), 'y': (-3, 3)})
# передамо алгоритми початкові точки для спроб
bo.explore({'x': [-1, 3], 'y': [-2, 2]})
# також можемо задати діапазони пошуку
bo.initialize(
    {
        'target': [-1, -1],
        'x': [1, 1],
        'y': [0, 2]
    }
)
```

```
# після опису вхідних умов просимо алгоритм знайти оптимальні
параметри функції
bo.maximize(init_points=5, n_iter=15, kappa=2)
# оптимальне значення
print(bo.res['max'])
```

3.1 Дані для експериментів

Для проведення експериментів перевірки роботи підходу та порівнянь різних типів моделей та архітектур було використано публічний набір даних, відкритий для дослідження, під назвою “Safecast: Відкриті дані про стан навколишнього середовища для кожного”. [23]

Safecast - це глобальний науковий проект громадян, орієнтований на добровольців, який допомагає людям отримувати дані про їх оточення. Спільнота, що розвизває доступ до цього набору даних, вважає, що більш дані про стан навколишнього середовища мають бути доступні всім для створення нових прогресивних проектів на благо людства.

Safecast надає дані про стан радіації, температуру та якість повітря у всьому світі. Незалежність, прозорість та відкритість є ключем успіху та популярності даного ресурсу. Safecast був швидко визнаний в Японії та за кордоном як надійне та неупереджене джерело екологічної інформації, яку громадяни можуть використовувати при прийнятті рішень.

Ця база даних зростає дуже швидко. На даний момент, публічна версія для завантаження має об’єм 11 Гб - це приблизно 100 млн. записів. Загалом, база містить наступні дані:

- Captured Time,
- Latitude
- Longitude
- Value
- Unit
- Location Name

- Device ID
- MD5Sum
- Height
- Surface
- Radiation
- Uploaded Time
- Loader ID.

У даній роботі буде використано дві колонки для побудови індексу: числове поле - відбиток часу - **Captured Time**, яке відповідає за час зняття показів, та строкове поле - **MD5Sum**, яке дозволяє перевірити запис на цілісність.

Дані публічно доступні на сторінці [24].

Розглянемо розподіл даних у колонці **Captured Time.**

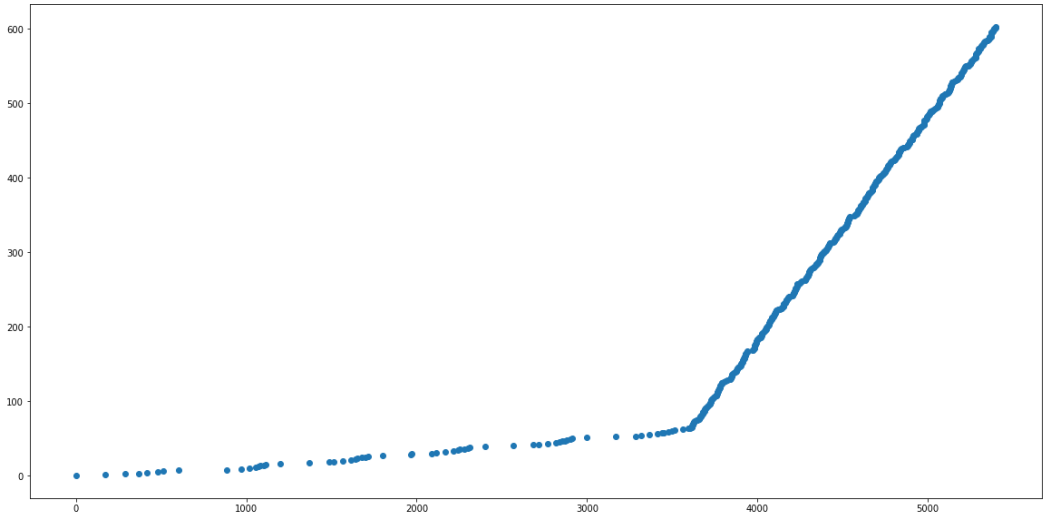

Рис. 3.4 — Розподіл даних у колонці Captured Time.

У такому масштабі функція виглядає як кусочно-лінійна. Однак, оглянемо розподіл детальніше - на рівні точок.

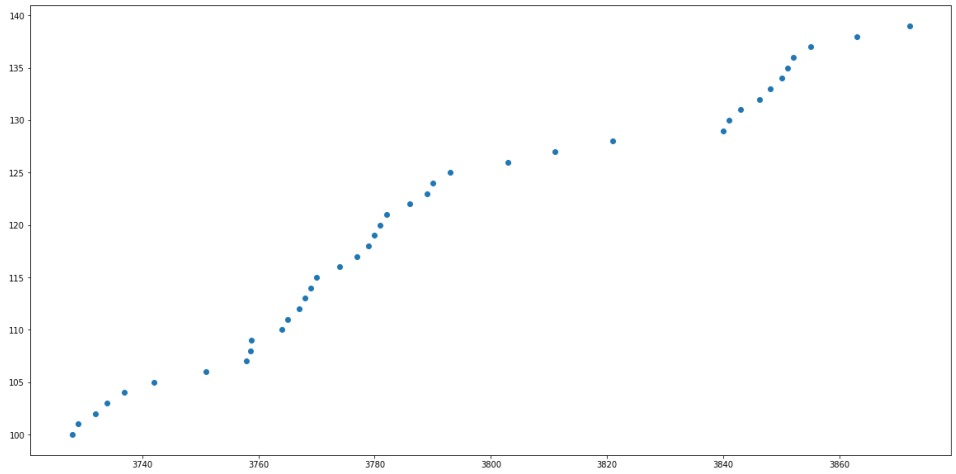

Рис. 3.5 — Розподіл даних у колонці Captured Time у великому масштабі.

Як бачимо, насправді функція розподілу **Captured Time** не слідує повністю лінійному закону.

Строкове поле **MD5Sum** містить MD5 хеш запису, наприклад:

33f029490e1fd6ecf7ed9bc60abad999

Традиційні підходи NLP не підходять для цього випадку, адже строкова колонка може містити не лише слова чи вирази, а й будь-якого виду токени, як наша колонка у базі. Найкращим методом цифрового представлення строк є перетворення їх у масив ASCII символів, у даному випадку - довжини 32.

Закодуємо наш попередній приклад:

[48, 48, 48, 50, 49, 53, 50, 51, 48, 98, 97, 52, 56, 98, 100, 56, 100, 99, 51, 53, 56, 55, 56, 102, 48, 100, 56, 57, 102, 51, 99, 54].

Якісно візуалізувати розподіл даних у вигляді функції 32 змінних важко, тому для оцінки якості моделі у строковому випадку обмежимось використанням графіку функції втрат.

3.2 Побудова та навчання моделі

У даній роботі була проведена низка експериментів для тестування можливості потенційного використання самокерованих індексних структур у базах даних. Перший завданням була перевірка теорій з базового дослідження [2]. Спочатку була побудована інфраструктура для досліджень та встановлено усі

необхідні бібліотеки, додатки та фреймворки, а також був отриманий доступ до ресурсів GPU Nvidia P5000. Також, у інфраструктуру були завантажені потрібні набори даних для зручності роботи з ними.

Розглянемо проведення експериментів поетапно.

3.2.1 Нормалізація даних

У алгоритмах машинного навчання головну роль грають дані та їх правильне представлення. Для того, щоб алгоритм навчався коректно та уникав *градієнтних вибухів*, дані варто нормалізувати. У глибинних мережах мережах градієнти помилок можуть накопичуватися під час оновлення та призвести до дуже великих градієнтів. Це, в свою чергу, призведе до великих оновлень ваг мережі та, в свою чергу, до їх нестабільності. В крайньому випадку значення ваг можуть стати настільки великими, що досягнуть ліміту розміру int32, і це призведе до значень NaN.

До прикладу, **Captured Time** містить числа такого порядку:

1525389999, 1525390170, 1525390285, 1525390367, 1525390415, 1525390479, 1525390513, 1525390599, 1525390885, 1525390967

Якщо подавати їх у такому вході у модель, вона муситиме підлаштовувати коефіцієнти таким чином, щоб такі великі входи трансформувати у різного порядку виходи. За таких умов навчити мережу доволі важко, адже оптимізатор може так і не знайти оптимального розв’язку. Тому для початку нормалізуємо дані.

Для числових значень, застосуємо наступну нормалізацію:

```
xs = data
xs = (xs - min(xs)) / max(xs)
```

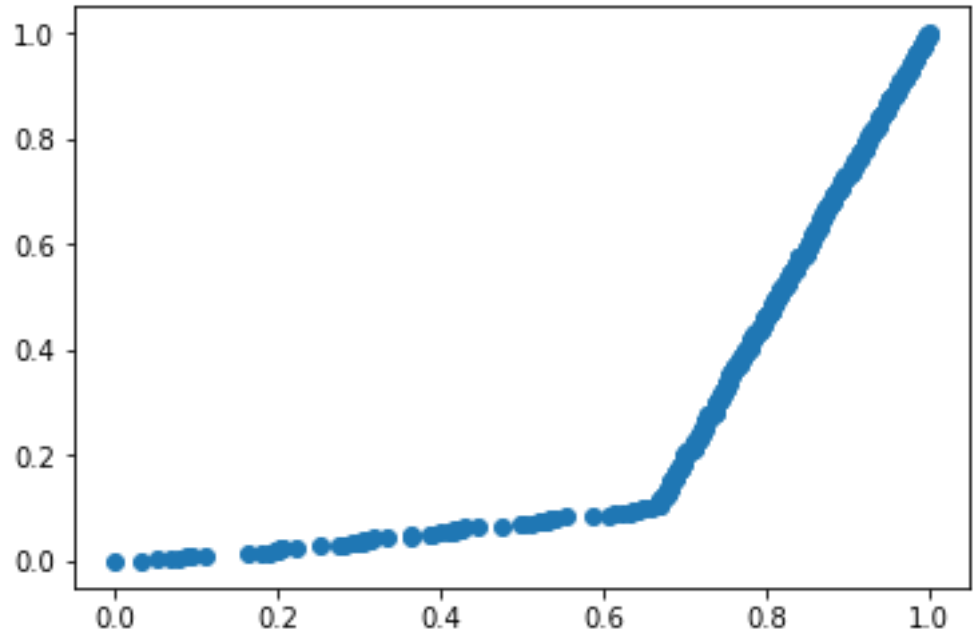

Рис. 3.6 — Розподіл нормалізованих даних у колонці Captured Time.

Так як на виході модель намагається передбачити позицію елемента у індексі - відсортованому масиві - вихідні значення будуть у проміжку 0..кількість елементів у `Xs`. Це також не дуже добре для моделі, адже розкид значень може бути дуже великим - від 0 до мільйонів, що також буде призводити до градієнтних вибухів та великих значень ваг. Тому нормалізуємо і вихідні значення:

```
Ys = outputs
Ys = Ys / len(outputs)
```

Таким чином, усі дані будуть входити у проміжок від 0 до 1.

Схожим чином подіємо на строкові дані. Як було згадано вище, у даній роботі будемо використовувати трансформацію строк у вектор ASCII символів, які мають діапазон 0...255. Також приведемо всі ці числа у проміжок від 0 до 1:

```
Xs = data
Xs = Xs / 255
```

Як і у для числових даних, аналогічно вчинимо і з виходами у цьому випадку.

```
Ys = outputs
Ys = Ys / len(outputs)
```

3.2.2 Побудова B-tree для отримання базових результатів

Для початку, побудуємо звичайний B-tree за допомогою Python пакету BTrees.

```
import torch as to
import pandas as pd
import datetime
import numpy as np

measurements_dataset_path = os.path.join(dataset_dir, 'measure-
ments.csv')
measurements_dataset = pd.read_csv(measurements_dataset_path,
nrows=5000000)
measurements_dataset['cap_time_timestamp'] = pd.to_datetime(measure-
ments_dataset['Captured Time'], format="%Y-%m-%d %H:%M:%S")
cleared_measurements_dataset = measurements_dataset[pd.isnull(measure-
ments_dataset['cap_time_timestamp']) == False].sort_val-
ues(by='cap_time_timestamp')
xs = np.unique(np.array(cleared_measurements_da-
taset['cap_time_timestamp'].map(pd.Timestamp.timestamp).map(float))[50
:-100])
ys = range(len(xs))

tree = BTrees.OOBTree.OOBTree()

for i, _ in enumerate(xs):
    tree.update({xs[i]: ys[i]})

list(t.values(min=xs[2000000], max=xs[2000001]))


md5sums = list(cleared_measurements_dataset.sort_val-
ues(by='MD5Sum')['MD5Sum'])
md5sums_ys = range(len(md5sums))
mtree = BTrees.OOBTree.OOBTree()

for i, _ in enumerate(md5sums):
    mtree.update({str(md5sums[i]): md5sums_ys[i]})

list(mtree.values(min=str(md5sums[20]), max=str(md5sums[21])))
```

Створимо два окремих дерева для чисельних та строкових даних і заміримо швидкість роботи на них. Результати можна знайти у таблиці 3.1.

Таблиця 3.1 — Результати побудови та пошуку даних у B-tree

Тип даних	Швидкість побудови. індексу	Швидкість роботи
Чисельні	14 с	6.1 мкс
Строкові	42 с	8.3 мкс

Бачимо, що індекс будується доволі швидко, а швидкість доступу до даних задовільна.

3.2.3 Побудова нейронної мережі плоскої архітектури

Як перевірка теорії, першою було побудовано кілька різновидів повнозв'язної нейронної мережі з двома прихованими шарами. Було проведено експерименти на частині набору цифрових та строкових даних.

У випадку чисельних даних, візьмемо 1000 перших міток даниз на вхід, а на виході очікуємо передбачення позиції у індексі розміром 1000. По суті, можемо розглядати це як задачу регресії та змусити мережу апроксимувати функцію розподілу даних.

```
Sequential(
 (0): Linear(in_features=1, out_features=50, bias=True)
   (1): ReLU()
   (2): Linear(in_features=50, out_features=50, bias=True)
   (3): ReLU()
   (4): Linear(in_features=50, out_features=50, bias=True)
   (5): ReLU()
   (6): Linear(in_features=50, out_features=1, bias=True)
 )
```

Використаємо середню квадратичну помилку як функцію втрат:

$$L(params) = \sum_i \left(y_i - \widehat{y_i}\right)^2 \tag{13}$$

Pytorch реалізовує її як об'єкт класу **to.nn.MSELoss**

```
loss_fn = to.nn.MSELoss(size_average=False)
```

Для того, щоб бачити більш повну картинку зміни функції втрат, не будемо використовувати усереднення по всім елементам батча.

Для оптимізації використаємо **стохастичний градієнтний спуск з імпульсом** та L2-регуляризацією у вигляді *weight decay*.

```
optimizer = to.optim.SGD(model.parameters(), lr=learning_rate,
momentum=0.9, weight_decay=1e-5)
```

Також для стійкості тренування було використано нелінійну активацію між шарами мережі у вигляді SELU, яка була описана у попередньому розділі роботи.

Pytorch використовує концепцію централізованого завантажувача даних - для цього реалізуємо інтерфейс класу Dataset:

```
class NumericalDataset(Dataset):
    def __init__(self, values, labels):
        self.values = values
        self.labels = labels
    def __len__(self):
        return len(self.values)
    def __getitem__(self, num):
        x = to.Tensor([self.values[num]])
        y = self.labels[num]
        sample = {'x': x, 'y': y}
        return sample
```

Клас NumericalDataset буде використовуватись завантажувачем даних для реалізації міні-батч підходу та рандомізації даних.

Для знаходження більш оптимального розв’язку, будемо використовувати розклад зміни коефіцієнту навчання - learning rate.

```
for learning_rate in [1e-3, 5e-4, 1e-4, 5e-5, 5e-6, 1e-6]:
        print(learning_rate)
        optimizer = to.optim.SGD(model.parameters(), lr=learning_rate,
momentum=0.9, weight_decay=1e-5)

        for epoch in range(epochs):
            model.train(True)
            for i, batch in enumerate(train_loader):
                x = Variable(cuda(batch['x']))
                y = Variable(cuda(batch['y'].type(to.FloatTen-
sor).view(-1, 1)))
                y_pred = model(x)
                loss = loss_fn(y_pred.view(-1, 1), y)
                if i % 100 == 0:
                    print(epoch, i, loss.data[0])
```

```
                    summary_writer.add_scalar(f'тренувальна помилка',
loss.data[0], epoch * train_total_batches + i)
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()
    model.train(False)
    print('загальна помилка на всьому наборі даних', float(loss_fn(
        model(cuda(to.Tensor([data_x]))),
        cuda(to.Tensor(data_y).type(to.FloatTensor).view(1, 1000, 1))
    ).data[0]))
```

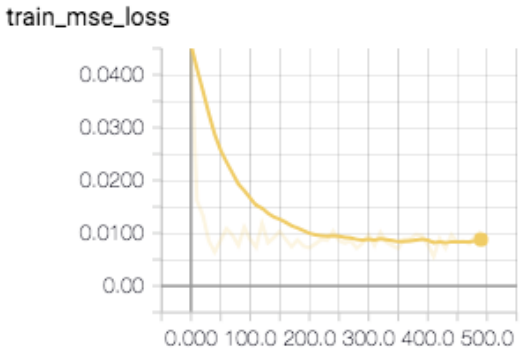

Рис. 3.7 — Графік поведінки функції помилки від часу тренування

За 500 ітерацій тренування ми досягли прийнятної помилки, зобразимо, як дана нейронна мережа апроксимувала наші дані.

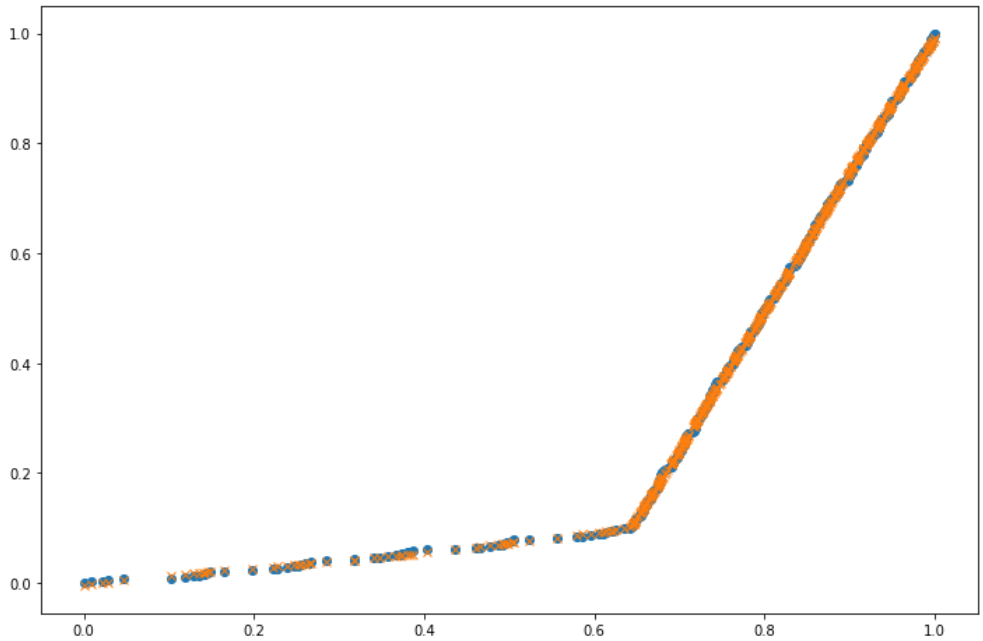

Рис. 3.8 — Накладення вивченого розподілу нормалізованих даних на реальний.

Як бачимо, наша модель змогла навчитись загальному розподілу даних, але локально можемо помітити доволі значну помилку.

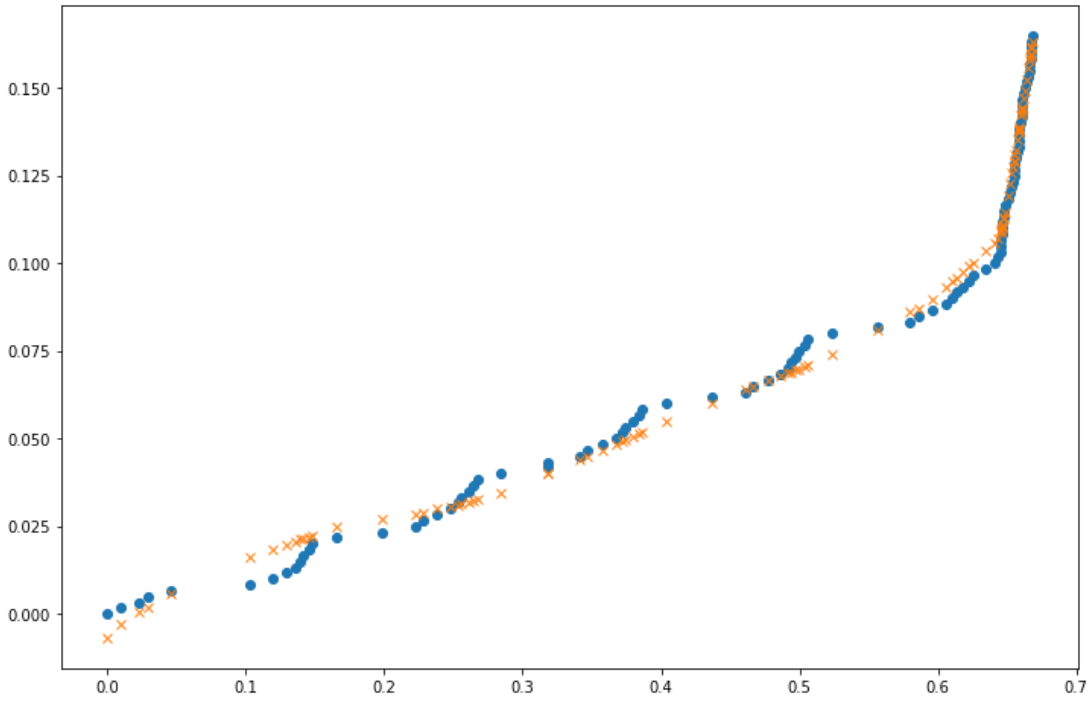

Рис. 3.9 — Накладення вивченого розподілу нормалізованих даних на реальний на великому масштабі.

Підрахуємо помилку на усьому датасеті та обчислимо максимальний розмір помилки, відповідно визначивши, скільки часу асимптотично потрібно для знаходження точно правильної відповіді:

```
tests_x = []
for i, test_x in enumerate(xs_n):
    tests_x.append(float(__loss_fn(
        mdl(cuda(to.Tensor([test_x]).view(-1, 1))),
        cuda(to.Tensor([ys_n[i]]).type(to.FloatTensor).view(-1, 1))
    ).data[0]))
```

Максимальна помилка на даному розмірі датасету: 0.00012313446495682, що відповідає помилці у 7 позицій, тобто для того, щоб гарантовано знайти елемент у індексі, потрібно переглянути 15 елементів, починаючи з передбаченої позиції: 7 зліва та 7 справа, що є доволі хорошим результатом.

Візьмемо датасет у 50000 записів. Його розподіл матиме інший вигляд:

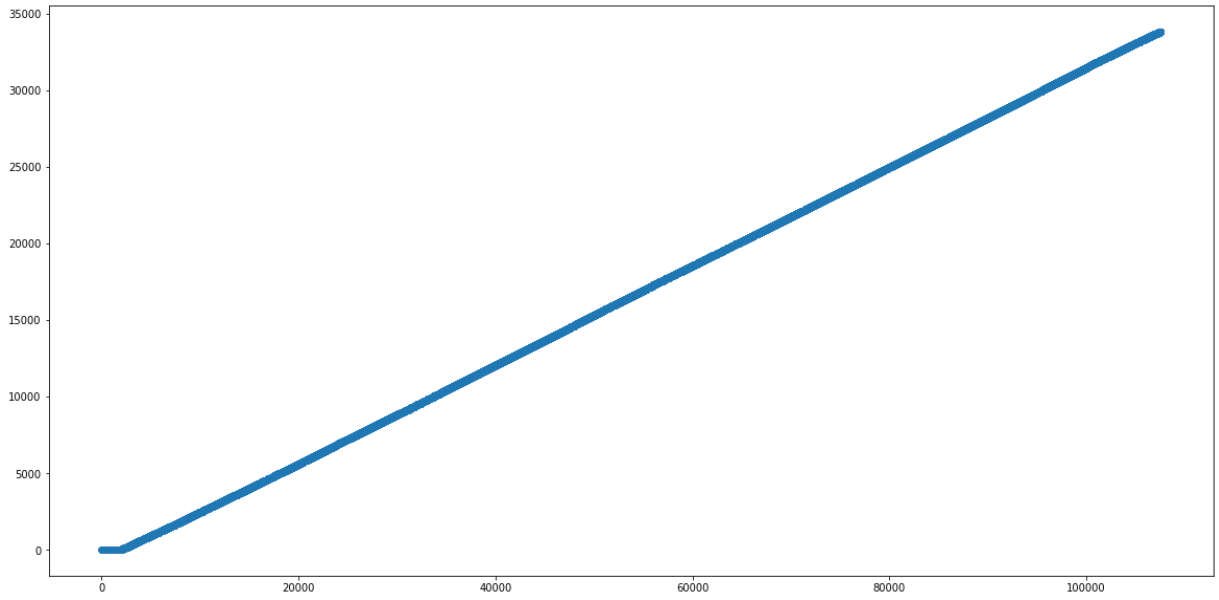

Рис. 3.10 — Розподіл даних на повному наборі даних

Після нормалізації даних

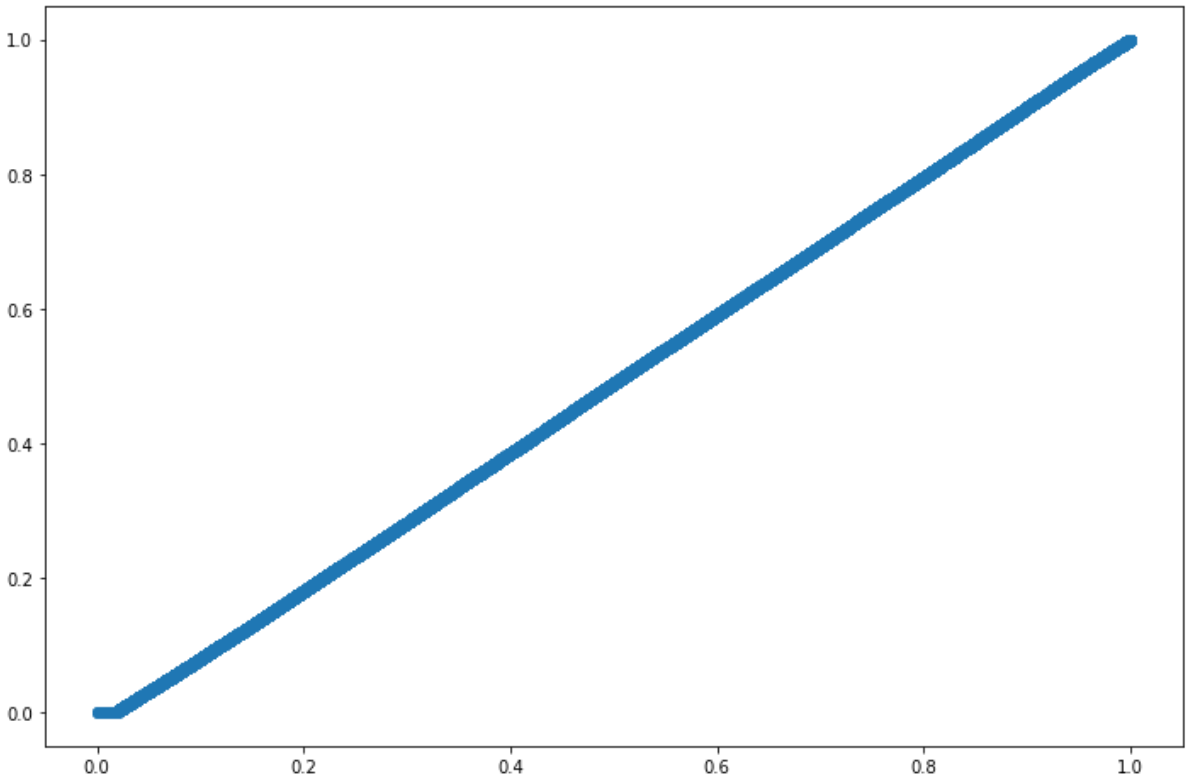

Рис. 3.11 — Розподіл нормалізованих даних на повному наборі даних

Поглянемо, як поводить себе натренована модель:

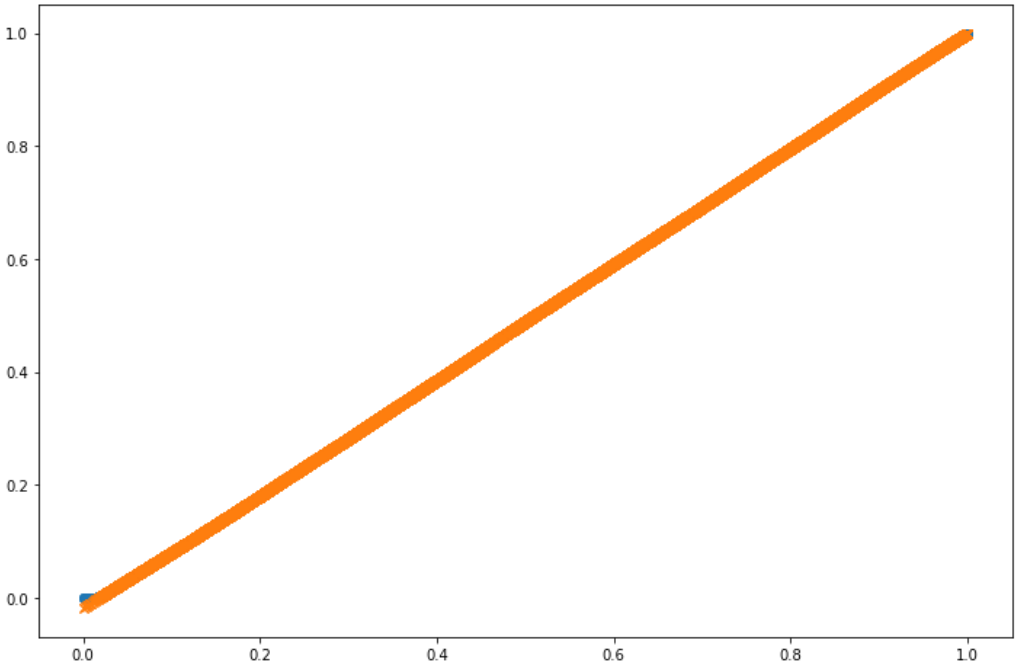

Рис. 3.12 — Накладення вивченого розподілу нормалізованих даних на реальний на повному наборі даних

Візуально зрозуміло, що найбільшою буде помилка на початку датасету - у даної моделі не вистачило потужності, щоб описати розподіл, а збільшення кількості параметрів зменшує ефективність моделі та все одно не дозволяє покращити точність (таблиця).

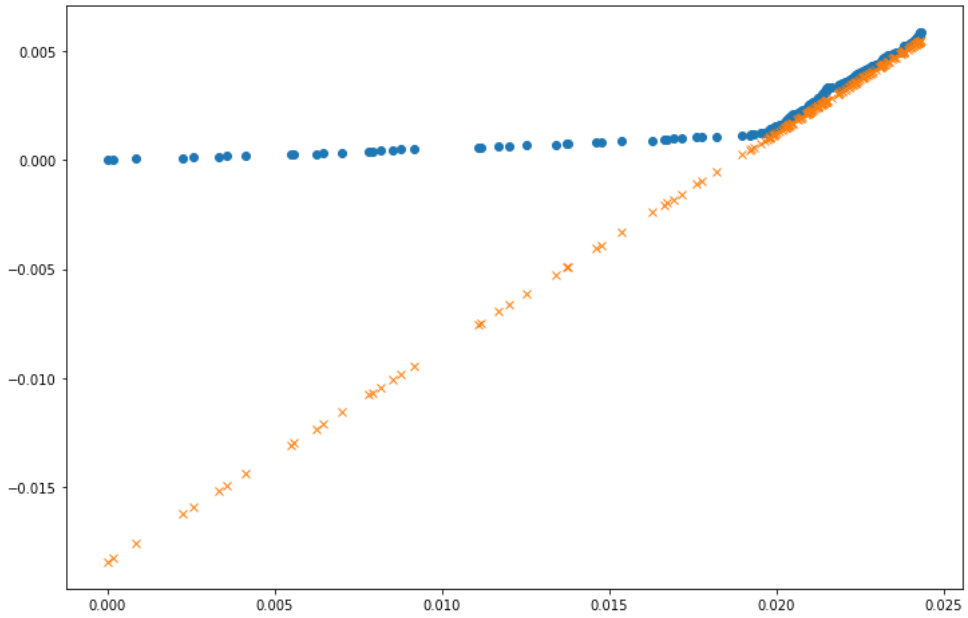

Рис. 3.13 — Накладення вивченого розподілу нормалізованих даних на реальний на повному наборі даних на великому масштабі

Поточна максимальна помилка - 623 позиції, тобто розмір віртуальної сторінки - 1047 елементів (623 вліво та 623 вправо), яких потрібно пройти, щоб гарантовано знайти елемент, що шукається, що є не надто гарним результатом в абсолютному сенсі.

Емпірично підібравши параметри та потестувавши інші моделі з різним розміров прихованих шарів, різною їх кількістю та різними активаціями, можна отримати дещо кращих результатів. У таблиці представлені найкращі результати для заданої тестової вибірки у 50000 позицій для даних **Captured Date.**

Варто зауважити, що швидкість роботи та швидкість тренувань було заміряна з проведенням обчислень на GPU.

Таблиця 3.2 — Результати експериментів різного типів моделей нейромереж для числових даних

Кількість прихованих шарів	Тип активації між шарами	Розмір прихованого шару	Швидкість тренування	Швидкість роботи	**Максимальна помилка**
2	SELU	100	192 с	1 мкс	**623**
3	SELU	100	284 с	1.3 мкс	**578**
3	ELU	100	299 с	2 мкс	**602**
5	SELU	80	367 с	4.5 мкс	**520**

Як бачимо, нарощення складності мережі дає покращення, але в той же час ми втрачаємо у швидкості, однак цього покращення недостатньо для такого погіршення у швидкості. Також найкраще серед усіх активацій проявила себе SELU, що показує важливість самонормалізації вагів.

Однак, якщо згадати, що помилка на датасеті розміром 1000 елементів давала нам похибку близько 10 позицій, стає зрозуміло, що разом зі збільшенням кількості елементів навіть за умов найсклкднішої мережі отримаємо доволі велику похибку.

Проведені тести для строкових даних показали навіть кращі результати: максимальна помилка у 207 позицій на мережі з двома прихованими шарами та SELU активацією. Порівняльна таблиця результатів 3.3.

Таблиця 3.3 — Результати експериментів різного типів моделей нейромереж для строкових даних

Кількість прихованих шарів	Тип активації між шарами	Розмір прихованого шару	Швидкість тренування	Швидкість роботи	**Максимальна помилка**
2	SELU	100	367 с	2 мкс	**207**
3	SELU	100	403 с	3.3 мкс	**192**
3	ELU	100	299 с	5 мкс	**204**
5	SELU	80	367 с	7 мкс	**178**

Зрозуміло, що у обох випадках абсолютна максимальна помилка все ще доволі велика, але проведенням цих експериментів підтверджено результати попередніх досліджень щодо якості передбачення моделі на “останній милі”, та як покращення було обрано рух у напрямку побудови моделі типу “гурт експертів”.

3.2.4 Побудова ієрархічної моделі

Гурт експертів — це техніка машинного навчання, в якій багато експертів (“учнів”, які навчаються на даних) використовуються для поділу простору вхідних даних на однорідні регіони [25]. Прикладом з домену комп’ютерного зору є поєднання однієї моделі нейронної мережі для детекції людини з іншою моделлю для оцінки пози. Якщо вивід з моделі базується на декількох рівнях імовірнісних функцій вибору моделі нижчого рівня, або так званих *гейтінг-функціях*, гурт називається ієрархічним гуртом експертів. [25]

Гейтінг-функція вирішує, який експерт використовувати для кожного регіону вводу. Навчання таким чином складається з 1) вивчення параметрів окремих учнів та 2) вивчення параметрів гейтінг функції.

У нашому випадку, для тесту побудуємо модель, котра має гейтинг-функцію та модель на останньому кроці, яка матиме по 1000 елементів. Тобто, маючи 50000 записів для тесту, протестуємо модель, яка матиме архітектуру наступного вигляду, де гейтинг-функція визначає одну з 50 моделей для передбачення своєї частини даних:

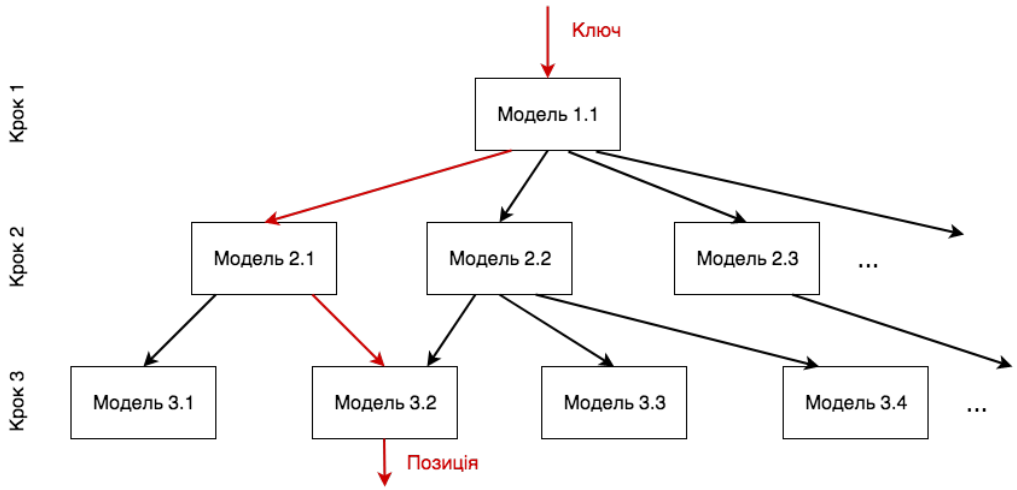

Рис. 3.14 — Запропонована модель типу “гурт експертів”

```
start = 0
finish = 49999
data_x = xs_n[start:finish]
data_y = ys_n[start:finish]
levels = [1, 10, 20]
models = []
for level in levels:
    num_samples_for_model = int(len(data_x) / level)
    tmp_models = []
    for i in range(level):
        print(level, i)
    tmp_models.append(train_mixture_model(data_x[i*num_sam-
ples_for_model:(i+1)*num_samples_for_model], data_y[i*num_sam-
ples_for_model:(i+1)*num_samples_for_model]))
    models.append(tmp_models)
```

На рис. 3.15 помітно, що модель почала описувати дані значно краще.

Однак, додавання ще одного рівня гейтинг-функцій дасть нам приріст у точності, хоч ми і трохи програємо у швидкості. У даній моделі максимальна

помилка - 87, тобто ми нам достатньо пройти 175 елементів для того, щоб гарантовано знайти елемент, що шукають. Однак, замість використання на останній милі нейромережі, можна використовувати простіше та легші у тренуванні методи, такі як метод опорних векторів.

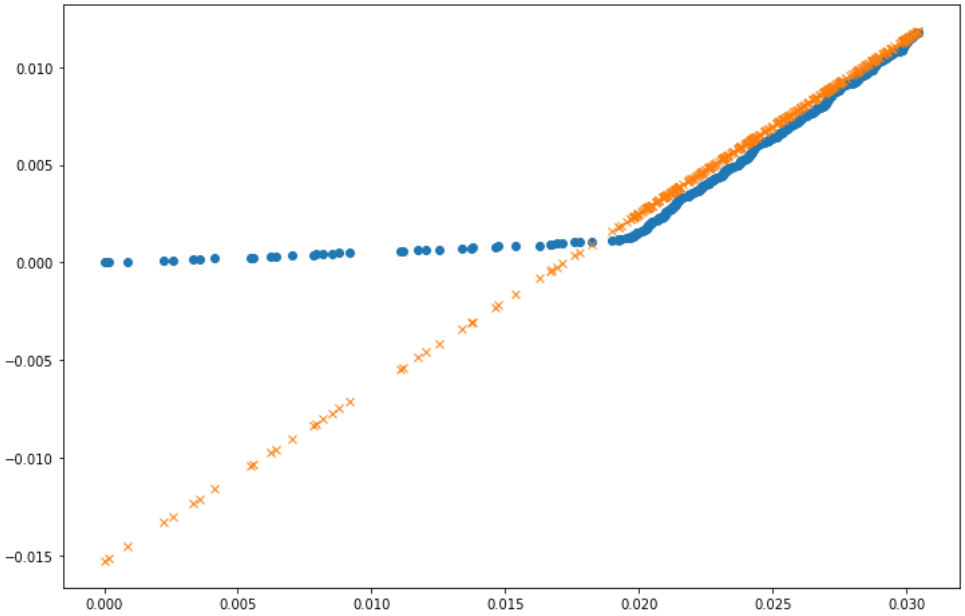

Рис. 3.15 — Покращення результатів: накладення вивченого розподілу нормалізованих даних на реальний на повному наборі даних на великому масштабі

Таблиця 3.4 — Результати експериментів з додавання моделі “гурт експертів”

Кількість рівнів гейтинг-функцій	Кількість прихованих шарів	Тип активації між шарами	Швидкість тренування	Швидкість роботи	**Максимальна помилка**
1	2	SELU	192 с	3 мкс	**124**
1	3	SELU	284 с	3.8 мкс	**96**
1	3	ELU	299 с	5.2 мкс	**116**
2	3	SELU	667 с	8.1 мкс	**59**

На таблиці 3.4 вище показано результати експериментів, які доводять, що додавання гейтинг-рівнів сильно додає у точності моделі.

3.2.5 Побудова моделі регресії за допомогою методу опорних векторів

Замість моделі передбачення у вигляді нейронної мережі на останньому рівні можна використати регресію за допомогою методу опорних векторів, і вона може давати приріст, якщо дані підпадають під природу певного ядра SVM.

Архітектура моделі залишиться такою ж, тільки тепер замість використання лише нейронної мережі на останній милі ми будемо тренувати алгоритм регресії методу опорних векторів та порівнювати з результатом нейронної мережі.

```
from sklearn import svm
clf = svm.SVR('poly', C=2.0)

clf.fit(xs_n.reshape(-1, 1), ys_n.reshape(-1, 1))
clf.predict(xs_n[10]) * np.max(ys)

ys_srv = clf.predict(xs_n.reshape(-1,1))
plt.plot(xs_n, ys_n, 'o')
plt.plot(xs_n, ys_srv, 'x')
plt.show()
```

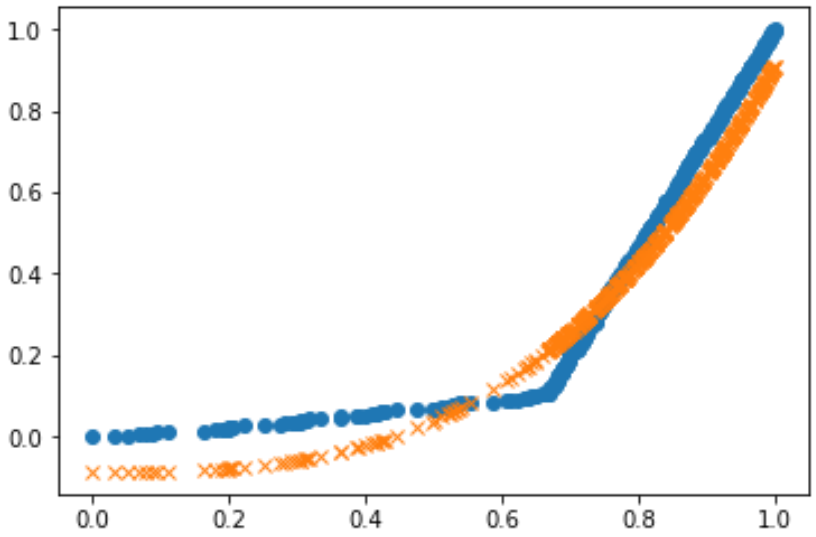

Рис. 3.15 — Результат роботи методи опорних векторів

Таблиця 3.5 — Результати експериментів з додавання моделі “гурт експертів” та варіаціями у типі моделі останнього рівня

Тип передбачення на останній милі	Кількість рівнів гейтинг-функцій	Швидкість тренування	Швидкість роботи	**Максимальна помилка**
Нейромережа	1	192 с	3 мкс	**124**
SVM	1	114 с	1.2 мкс	**205**
Автоматичний вибір між кращим алгоритмомо	1	269 с	2.5 мкс	**116**
Нейромережа	2	667 с	6.3 мкс	**59**
SVM	2	416 с	3.8 мкс	**143**
Автоматичний вибір між кращим алгоритмомо	2	840 с	5.2 мкс	**23**

Бачимо, що на останній лінії помилка з використанням SVM у даному випадку зросла, однак він чудово підходить у випадках, коли дані розподілені дуже нелінійно, однак потрібно передбачити випадки, коли його краще застосовувати. Використання SVM чи нейромережі на останній милі - також один із гіперпараметрів цілісної моделі, тож далі нам потрібен механізм підобру гіперпараметрів для підвищення точності.

3.2.6 Використання пакету **Bayesian Optimization** для пошуку найкращих гіперпараметрів

Для застосування Байєсівської оптимізації, опишемо функцію, яка буде використуватись для мінімізації значення. У даному випадку, ми хочемо мінімізувати функцію помилок для моделі на даному наборі даних.

Для цього загорнемо функцію тренування моделі у функцію `optimize`. Пакет **Bayesian Optimization** самостійно вміє лише максимізувати функцію, але у нашому випадку це не проблема, адже максимізація функції - це мінімізації від'ємної функції.

Спочатку пакет зробить 10 початкових вимірів і отримає перше приближення функції помилки. Після цього за допомогою Гаусівських процесів почнеться процес мінімізації.

```
bo = bayes_opt.BayesianOptimization(optimize, {
    'hidden_size':(5, 64),
    'batch_size':(4,32),
    'activation_idx':(0,4),
    'dropout_prob':(0.3,0.7),
    'layers_num': (1,4),
    'weight_decay':(1e-5, 1e-4),
    'learning_rate': (1e-5, 1e-2),
    'layers': (1, 3)
})
bo.maximize(n_iter=20)
```

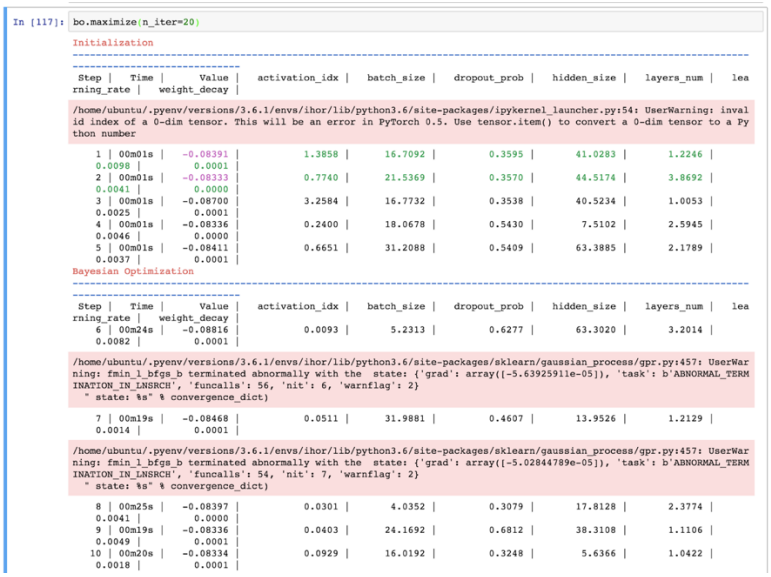

Рис. 3.15 — Приклад роботи пакету Bayesian Optimization: фази ініціалізації та підбору

У результаті, отримані гіперпараметри змогли покращити поточні моделі.

Таблиця 3.6 — Результати експериментів з використанням Байєсівської оптимізації гіперпараметрів

Тип передбачення на останній милі	Кількість рівнів гейтинг-функцій	Швидкість тренування	Швидкість роботи	**Максимальна помилка**
Нейромережа	1	1650 с	3 мкс	**87**
SVM	1	114 с	1.2 мкс	**183**
Автоматичний вибір між кращим алгоритмомо	1	269 с	2.5 мкс	**98**
Нейромережа	2	667 с	6.3 мкс	**29**
SVM	2	416 с	3.8 мкс	**114**
Автоматичний вибір між кращим алгоритмомо	2	840 с	5.2 мкс	**23**

3.3 Висновки

Отже, в даному розділі було описано деталі реалізації самоналагоджувальних індексних структур для діапазонного пошуку, а також описано проведення експериментів, що показали ефективність використання таких структур.

Для початку було проведено аналіз існуючих середовищ розробки та фреймворків для роботи з алгоритмами машинного навчання. В результі цього було обрано середовище програмування Juputer з використанням мови програмування Python, а також фреймворк для побудови моделей глибокого навчання PyTorch та класичних методів машинного навчання Scikit-learn.

Було показано деталі реалізації трьої видів самоналагоджувальних індексних структур для діапазонного пошуку: на базі використання неглибоких нейронних мереж, на базі використання методу регресії опорних векторів. Результати експериментів показали наявність проблеми передбачення на “останній милі”, тобто модель гарно вивчає загальний розподіл даних, але може доволі сильно помилятись при визначенні точної позиції. Для зменшення випливу цієї проблемибули застосувані моделі типу гурт експертів, які дозволяють суттєво покращити результати. У рамках даної моделі було зроблено порівняння моделей у вигляді нейронних мереж та методу регресії опорних векторів, які показали схожі результати в залежності від набору даних. Для покращення останнього типу моделей також було застосовано автоматичний підбір гіперпараметрів на основі методів Байєсівської оптимізації, що дозволило покращити результати роботи моделі.

Було наведено результати експериментів по використанню запропонованих самоналагоджувальних індесних структур на різних наборах даних: чисельних та строкових. Ці результати показали придатність використання навчених індексних структур та перспективу їх подальшого дослідження та інтеграції у бази даних або сторонні системи індексації.

4 РЕАЛІЗАЦІЯ СТАРТАП-ПРОЕКТУ

4.1 Опис ідеї та технологічний аудит стартап-проекту

У даному розділі описано економічне обґрунтування реалізації стартап-проекту на тему «Створення системи керування базами даних з адаптивними індексними структурами».

Таблиця 4.1 – Опис ідеї стартап-проекту

Зміст ідеї	*Напрямки застосування*	*Вигоди для користувача*
Ідея полягає у тому, щоб створити СКБД, в якій будуть використовуватись адаптивні індексні структури даних для швидкого пошуку на основі підходів машинного навчання	1. Вирішення задачі швидкого доступу до даних	Користувачу необхідно буде встановити СКБД, створити базу даних і додати в неї свої дані. Після цього в нього буде можливість швидкого доступу до даних
	2. Вирішення задачі адаптування індексів до конкретних розподілів даних	В користувача буде можливість ефективніше використовувати обчислювальні ресурси внаслідок використання у базі даних адаптивних індексних структур

Аналіз потенційних техніко-економічних переваг ідеї порівняно із пропозиціями конкурентів передбачає:

1. визначення переліку техніко-економічних властивостей та характеристик ідеї
2. визначення попереднього кола конкурентів (проектів-конкурентів) або товарів-замінників чи товарів-аналогів, що вже існують на ринку, та проводиться збір

інформації щодо значень техніко-економічних показників для ідеї власного проекту та проектів-конкурентів відповідно до визначеного вище переліку;

3. проводиться порівняльний аналіз показників: для власної ідеї визначаються показники, що мають а) гірші значення (W, слабкі); б) аналогічні (N, нейтральні) значення; в) кращі значення (S, сильні).

Таблиця 4.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

No п/п	*Техніко-економічні характеристики ідеї*	*(потенційні) товари/концепції конкурентів*				*W (слабка сторона)*	*N (нейтральна сторона)*	*S (сильна сторона)*
		Мій проект	*Конкурент1*	*Конкурент2*	*Конкурент3*			
1.	Форма виконання	Програма	Програма	Веб-додаток	Програма			+
2.	Собівартість	Низька	Висока	Низька	Висока			+
3.	Наявність адміністратора для налаштування	Треба	Не треба, дистанційно	Треба	Треба		+	
4.	Наявність інтернету	Не треба	Не треба	Треба	Не треба			+
5.	Кросплатформеність	Так	Так	Так	Ні	+		

Визначений перелік слабких, сильних та нейтральних характеристик та властивостей ідеї потенційного товару є підґрунтям для формування його конкурентоспроможності.

В межах даного підрозділу необхідно провести аудит технології, за допомогою якої можна реалізувати ідею проекту (технології створення товару).

Визначення технологічної здійсненності ідеї проекту передбачає аналіз таких складових:

1. за якою технологією буде виготовлено товар згідно ідеї проекту?
2. Чи існують такі технології,чи їх потрібно розробити/доробити?
3. чи доступні такі технології авторам проекту?

Таблиця 4.3 – Технологічна здійсненність ідеї проекту

No п/п	*Ідея проекту*	*Технології її реалізації*	*Наявність технологій*	*Доступність технологій*
1.	Створення нової СКБД	Amazon Web Services	Наявна	Платна, недоступна
2.		C/C++, TensorFlow	Наявна	Безкоштовна, доступна
Обрана технологія реалізації ідеї проекту: для створення системи керування базою даних даних обрано технології C/C++ TensorFlow, яка є безкоштовною та доступною.				

4.2 Аналіз ринкових можливостей

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів.

Спочатку проводимо аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку.

Таблиця 4.4 – Попередня характеристика потенційного ринку стартап-проекту

No n/ n	*Показники стану ринку (найменування)*	*Характеристика*
1	Кількість головних гравців, од	3
2	Загальний обсяг продаж, грн/ум.од	20000 грн./ум.од
3	Динаміка ринку (якісна оцінка)	Зростає/спадає/стагнує
4	Наявність обмежень для входу (вказати характер обмежень)	Немає
5	Специфічні вимоги до стандартизації та сертифікації	Немає
6	Середня норма рентабельності в галузі (або по ринку), %	R = (3000000 * 100) / (1000000 * 12) = 25%

Рентабельність— поняття, що характеризує економічну ефективність виробництва, за якої за рахунок грошової виручки від реалізації продукції (робіт, послуг) повністю відшкодовує витрати на її виробництво й одержується прибуток як головне джерело розширеного відтворення [http://buklib.net/books/29473/]. Суть одного із найважливіших методів оцінки економічної ефективності інвестицій полягає у розрахунку їх середньої рентабельності за формулою [http://pidruchniki.com/1566072162240/turizm/prognozuvannya_efektivnosti_investi tsiynogo_proektu]

$$R = \frac{P}{1 \times n} \times 100$$

де Р- прибуток за час експлуатації проекту; / - повна сума інвестиційних витрат; п - час експлуатації проекту. Інвестувати грошові засоби доцільно тоді, коли від цього можна отримати більший прибуток, ніж від їх зберігання у банку. Порівнюючи

середньорічну рентабельність інвестицій зі ставкою банківського відсотка, можна дійти висновку, що вигідніше [26].

Таблиця 4.5 – Характеристика потенційних клієнтів стартап-проекту

No n/n	*Потреба, що формує ринок*	*Цільова аудиторія (цільові сегменти ринку)*	*Відмінності у поведінці різних потенційних цільових груп клієнтів*	*Вимоги споживачів до товару*
1.	Необхідно програмне забезпечення для обробки великих наборів даних і швидкого доступу до них	Потенційними цільовими групами є дослідницькі центри, університети та компанії, специфіка роботи яких пов‘язана із великими даними	Цільова група займається дослідженнями або має обробляти даних великих розмірів	Рішення має бути швидким, що дозволить концентруватись на інших задачах, і використовувати даний програмний продукт як інструментарій

Після визначення потенційних груп клієнтів проводиться аналіз ринкового середовища: складаються таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (табл. NoNo 6-7). Фактори в таблиці подавати в порядку зменшення значущості.

Ринкові можливості - це сприятливі обставини, які підприємство може використовувати для отримання переваг. Слід зазначити, що можливостями з

погляду SWOT-аналізу є не всі можливості, які існують на ринку, а тільки ті, які можна використовувати. [27]

Ринкові загрози - події, настання яких може несприятливо вплинути на підприємство. [28]

Таблиця 4.6 – Фактори загроз

No n/n	*Фактор*	*Зміст загрози*	*Можлива реакція компанії*
1.	Конкуренція	Вихід на ринок великої компанії	1) Вихід з ринку 2) Запропонувати великій компанії поглинути себе 3) Передбачити додаткові переваги власного ПЗ для того, щоб повідомити про них саме після виходу міжнародної компанії на ринок
2.	Зміна потреб користувачів	Користувачам необхідне програмне забезпечення з іншим функціоналом	1) Передбачити можливість додавання нового функціоналу до створюваного ПЗ

Таблиця 4.7 – Фактори можливостей

No n/n	*Фактор*	*Зміст можливості*	*Можлива реакція компанії*
1	Зростання можливостей потенційних покупців	Зростанням держфінансувауня досліджень у галузі обробки великих наборів даних	Запропонувати свої послуги державним підприємствам та дослідницьким центрам
2	Зниження довіри до конкурента 1	У ПЗ конкурента No1 нещодавно була знайдена помилка, завдяки якій дані досліджень усіх клієнтів стали доступні в інтернеті для всіх користувачів	При виході на ринок звертати увагу покупців на безпеку нашого ПЗ

Надалі проводиться аналіз пропозиції: визначаються загальні риси конкуренції на ринку (табл. 4.8).

Таблиця 4.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	*В чому проявляється дана характеристика*	*Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)*
1. Вказати тип конкуренції - досконала	Існує 3 фірми-конкурентки на ринку	Врахувати ціни конкурентних компаній на початкових етапах створення бізнесу, реклама (вказати на конкретні переваги перед конкурентами)
2. За рівнем конкурентної боротьби - міжнародний	Одна з компаній – з ішної країни, дві – з України	Додати можливість вибору мови ПЗ, щоб легше було у майбутньому вийти на міжнародний ринок
3. За галузевою ознакою - внутрішньогалузева	Конкуренти мають ПЗ, яке використовується лише всередині даної галузі	Створити основу ПЗ таким чином, щоб можна було легко переробити дане ПЗ для використання у інших галузях
4. Конкуренція за видами товарів: - товарно-видова	Види товарів є однаковими, а саме – програмне забезпечення	Створити ПЗ, враховуючи недоліки конкурентів
5. За характером конкурентних переваг неціновa	Вдосконалення технології створення ПЗ, щоб собівартість була нижчою	Використання менш дорогих технологій для розробки, ніж

		використовують конкуренти
6. За інтенсивністю - не марочна	Бренди відсутні	-

Після аналізу конкуренції проводиться більш детальний аналіз умов конкуренції в галузі. *М. Портер* вирізняє п‘ять основних факторів, що впливають на привабливість вибору ринку з огляду на характер конкуренції.

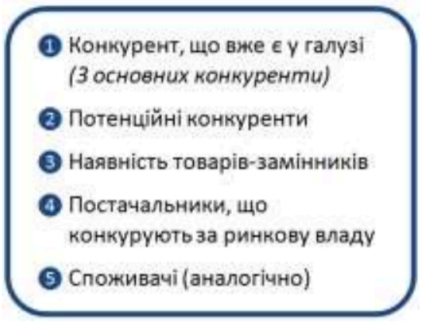

Рис. 4.1 – Складові моделі 5 сил М. Портера

Сильні позиції компанії за кожним з факторів означають її можливості забезпечити необхідні темпи обороту капіталу та її здатність впливати на інших агентів ринку, диктуючі їм власні умови співпраці.

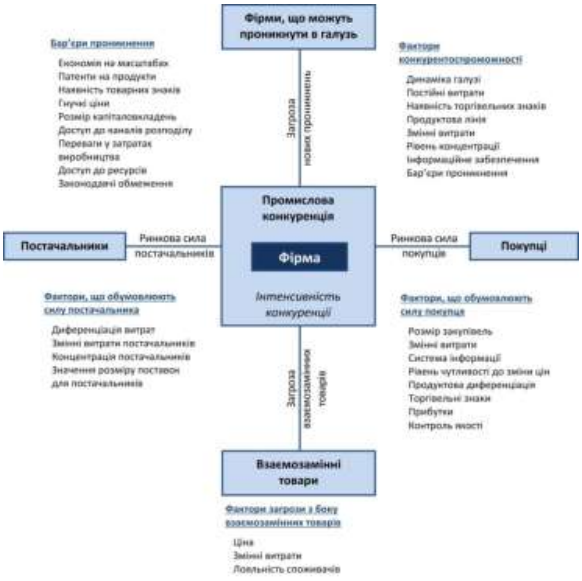

Рис. 4.2 – Модель 5 сил М. Портера для аналізу конкуренції в галузі

Характеристики факторів моделі відрізняються для різних галузей та змінюються із часом. Сила кожного фактору є функцією від структури галузі та її техніко-економічних характеристик.

На основі аналізу складових моделі 5 сил М. Портера розробляється перелік факторів конкурентоспроможності для певного ринку.

Таблиця 4.9 – Аналіз конкуренції в галузі за М. Портером

	Прямі конкуренти в галузі	*Потенційні конкуренти*	*Постачальники*	*Клієнти*	*Товари-замінники*
Складові аналізу	*Навести перелік прямих конкурентів*	*Визначити бар'єри входження в ринок*	*Визначити фактори сили постачальників*	*Визначити фактори сили споживачів*	*Фактори загроз з боку замінників*
Висновки:	Існує 3 конкуренти на ринку. Найбільш схожим за виконанням є конкурент 1, так як його рішення також представлене у вигляді програми.	Так, можливості для входу на ринок є, бо наше рішення спрощує та пришвидшує роботу спеціаліста.	Постачальники відсутні.	Важливим для користувача є швидкість роботи ПЗ.	Товари-замінники можуть використати більш дешеву технологію створення ПЗ та зменшити собівартість товару.

За результатами аналізу таблиці робиться висновок щодо принципової можливості роботи на ринку з огляду на конкурентну ситуацію. Також робиться висновок щодо характеристик (сильних сторін), які повинен мати проект, щоб бути конкурентоспроможним на ринку. Другий висновок враховується при формулюванні переліку факторів конкурентоспроможності у п. 3.6.

На основі аналізу конкуренції, проведеного в п. 3.5 (табл. 9), а також із урахуванням характеристик ідеї проекту (табл. 2), вимог споживачів до товару (табл. 5) та факторів маркетингового середовища (табл. NoNo 6-7) визначається та обґрунтовується перелік факторів конкурентоспроможності. Аналіз оформлюється за табл. 10.

Таблиця 4.10 – Обґрунтування факторів конкурентоспроможності

No n/n	*Фактор конкурентоспроможності*	*Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)*
1.	Виконання ПЗ у вигляді програми у розподіленій системі	Це рішення дозволяє швидко обробляти дані розміром значно більше за 4 ГБ.
2.	Простота інтерфейсу користувача	Користувач має лише завантажити дані і запустити програму на виконання.

За визначеними факторами конкурентоспроможності (табл. 10) проводиться аналіз сильних та слабких сторін стартап-проекту (табл. 11). Фінальним етапом ринкового аналізу можливостей впровадження проекту є складання SWOT-аналізу (матриці аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities) на основі виділених ринкових загроз та можливостей, та сильних і слабких сторін.

Таблиця 4.11 – Порівняльний аналіз сильних та слабких сторін проекту

No n/ n	*Фактор конкурентоспроможності*	*Бали 1-20*	*Рейтинг товарів-конкурентів у порівнянні з нашим підприємством*						
			–3	*–2*	*–1*	*0*	*+1*	*+2*	*+3*
1	Виконання ПЗ у вигляді програми у розподіленій системі	15			+				
2	Простота інтерфейсу користувача	20	+						

Перелік ринкових загроз та ринкових можливостей складається на основі аналізу факторів загроз та факторів можливостей маркетингового середовища. Ринкові загрози та ринкові можливості є наслідками (прогнозованими результатами) впливу факторів, і, на відміну від них, ще не є реалізованими на ринку та мають певну ймовірність здійснення. Наприклад: зниження доходів потенційних споживачів – фактор загрози, на основі якого можна зробити прогноз щодо посилення значущості цінового фактору при виборі товару та відповідно, – цінової конкуренції (а це вже – ринкова загроза).

Таблиця 4.12 – SWOT-аналіз стартап-проекту

Сильні сторони: простий інтерфейс користувача, виконання ПЗ у вигляді програми	Слабкі сторони: необхідно мати власний сервер (або орендувати його)
Можливості: у конкурента 1 виявлена проблема із безпекою ПЗ, додаткове держфінансування для досліджень у підприємствах, які є потенційними покупцями	Загрози: конкуренція, зміна потреб користувачів

На основі SWOT-аналізу розробляються альтернативи ринкової поведінки (перелік заходів) для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок. Визначені альтернативи аналізуються з точки зору строків та ймовірності отримання ресурсів.

Таблиця 4.13 – Альтернативи ринкового впровадження стартап-проекту

No n/n	*Альтернатива (орієнтовний комплекс заходів) ринкової поведінки*	*Ймовірність отримання ресурсів*	*Строки реалізації*
1.	Створення доповнення до існуючої СКБД, наприклад PostgresSQL	80%	6 місяців
2.	Створення програми без використання будь- яких фреймворків для обробки даних	30%	12 місяців

З означених альтернатив обирається та, для якої: а) отримання ресурсів є більш простим та ймовірним; б) строки реалізації – більш стислими. Тому обираємо альтернативу 1.

4.3 Розробка ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів.

Таблиця 4.14 – Вибір цільових груп потенційних споживачів

No п/п	*Опис профілю цільової групи потенційних клієнтів*	*Готовність споживачів сприйняти продукт*	*Орієнтовний попит в межах цільової групи (сегменту)*	*Інтенсивність конкуренції в сегменті*	*Простота входу у сегмент*
1.	Університетські дослідження	Час виконання не є критичним у даному випадку, а однією із найголовніших переваг є саме це	Дослідження в області великих даних проводяться постійно	Існує 3 конкуренти, які надають схожі, але менш швидкі рішення.	У сегмент увійти непросто, бо бюрократія всередині університеті в занадто складна, також відсутня дуже впливова перевага ПЗ

2.	Дослід-ницькі центри	Пришвидшення часу виконання додає їм можливості виконати більшу кількість замовлень	Дослідження в області великих даних проводяться постійно, до того ж – за замовлення м приватних компаній		Маючи перевагу у зручності інтерфесу користувача, що пришвидшує роботу та отримання результату, вийти на ринок нескладно
3.	Підприє - мства	Пришвидшення часу виконання додає їм можливості виконати більшу кількість замовлень, до того ж в цьому році з держбюджету виділені додаткові кошти на цей напрям досліджень			Маючи перевагу у тому, що рішення є зручним для користуванн я і швидким вийти на ринок не є складно
Які цільові групи обрано: обираємо дослідницькі центри та підприємства, які підтримуються держзамовленням					

За результатами аналізу потенційних груп споживачів (сегментів) автори ідеї обирають цільові групи, для яких вони пропонуватимуть свій товар, та визначають стратегію охоплення ринку.

Для роботи в обраних сегментах ринку необхідно сформувати базову стратегію розвитку. За М. Портером, існують три базові стратегії розвитку, що відрізняються за ступенем охоплення цільового ринку та типом конкурентної переваги, що має бути реалізована на ринку (за витратами або визначними якостями товару).

Стратегія лідерства по витратах передбачає, що компанія за рахунок чинників внутрішнього і/або зовнішнього середовища може забезпечити більшу, ніж у конкурентів маржу між собівартістю товару і середньоринковою ціною (або ж ціною головного конкурента).

Стратегія диференціації передбачає надання товару важливих з точки зору споживача відмітних властивостей, які роблять товар відмінним від товарів конкурентів. Така відмінність може базуватися на об‘єктивних або суб‘єктивних, відчутних і невідчутних властивостях товару(у ширшому розумінні – комплексі маркетингу), бути реальною або уявною. Інструментом реалізації стратегії диференціації є ринкове позиціонування.

Стратегія спеціалізації передбачає концентрацію на потребах одного цільового сегменту, без прагнення охопити увесь ринок. Мета тут полягає в задоволенні потреб вибраного цільового сегменту краще, ніж конкуренти. Така стратегія може спиратися як на диференціацію, так і на лідерство по витратах, або і на те, і на інше, але тільки у рамках цільового сегменту. Проте низька ринкова доля у разі невдалої реалізації стратегії може істотно підірвати конкурентоспроможність компанії.

Таблиця 4.15 – Визначення базової стратегії розвитку

No n/ n	*Обрана альтернатива розвитку проекту*	*Стратегія охоплення ринку*	*Ключові конкурентоспромо жні позиції відповідно до обраної альтернативи*	*Базова стратегія розвитку**
1.	Створення програми з використанням C/C++, Tensor-Flow	Ринкове позіціювання	Простота інтерфейсу, пришвидшення роботи	Диференціації

Наступним кроком є вибір стратегії конкурентної поведінки.

Стратегія лідера. Залежно від міри сформованості товарного(галузевого) ринку, характеру конкурентної боротьби компанії-лідери обирають одну з трьох стратегій: розширення первинного попиту, оборонну або наступальну стратегію або ж застосувати демаркетинг або диверсифікацію.

Стратегія розширення первинного попиту доцільна у разі, якщо фірмі-лідерові недоцільно розмінюватися на боротьбу з невеликими конкурентами, вона може отримати велику економічну віддачу від розширення первинного рівня попиту. В цьому випадку компанія займається реалізацією заходів по формуванню попиту(навчанню споживачів користуванню товаром, формування регулярного попиту, збільшення разового споживання), також пропаганду нових напрямів застосувань існуючих товарів, виявлень нових груш споживачів.

У міру зростання ринку, його становлення позиції компанії-новатора починають атакувати конкуренти-імітатори. В цьому випадку, компанія може вибрати оборонну стратегію, метою якої є захист власної ринкової долі.

Наступальна стратегія припускає збільшення своєї частки ринку. При цьому переслідувана мета полягає в подальшому підвищенні прибутковості роботи компанії на ринку за рахунок максимального використання ефекту масштабу.

Якщо фірма потрапляє під дію антимонопольного законодавства, вона може удатися до стратегії демаркетинга, що припускає скорочення своєї частки ринку, зниження рівня попиту на деяких сегментах за рахунок підвищення ціни. При цьому ставиться завдання недопущення на ці сегменти конкурентів, а компенсація втрат прибутку через зменшення обсягів виробництва компенсується встановленням надвисоких цін.

Стратегія виклику лідера. Стратегію виклику лідерові найчастіше вибирають компанії, які є другими, третіми на ринку, але бажають стати лідером ринку. Теоретично, ці компанії можуть прийняти два стратегічні рішення: атакувати лідера у боротьбі за частку ринку або ж йти за лідером.

Рішення атакувати лідера є досить ризикованим. Для його реалізації потрібні значні фінансові витрати, know – how, краще співвідношення «ціна- якість», переваги в системі розподілу і просування і т. д. У разі не реалізації цієї стратегії, компанія може бути відкинута на аутсайдерські позиції на досить довгий час. Залежно від цього компанія може вибрати одну з альтернативних стратегій: фронтальної або флангової атаки.

Стратегія наслідування лідеру. Компанії, що приймають слідування за лідером – це підприємства з невеликою часткою ринку, які вибирають адаптивну лінію поведінки на ринку, усвідомлюють своє місце на нім і йдуть у фарватері фірм-лідерів. Головна перевага такої стратегії – економія фінансових ресурсів, пов‘язаних з необхідністю розширення товарного(галузевого) ринку, постійними інноваціями, витратами на утримання домінуючого положення.

Стратегія заняття конкурентної ніші. При прийнятті стратегії зайняття конкурентної ніші (інші назви – стратегія фахівця або нішера) компанія в якості цільового ринку вибирає один або декілька ринкових сегментів. Головна особливість – малий розмір сегментів/сегменту. Ця конкурентна стратегія являється похідною від такої базової стратегії компанії, як концентрація. Головне завдання для

компаній, що вибирають стратегію нішера або фахівця, – це постійна турбота про підтримку і розвиток своєї конкурентної переваги, формування лояльності і прихильності споживачів, підтримка вхідних бар‘єрів.

Таблиця 4.16 – Визначення базової стратегії конкурентної поведінки

No п/п	*Чи є проект «першопрохідцем» на ринку?*	*Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?*	*Чи буде компанія копіювати основні характеристики товару конкурента, і які?*	*Стратегія конкурентної поведінки**
1.	Ні	Так	Буде, а саме: основною задачею ПЗ є зберігання даних і швидкий доступ до них (конкуренти 1, 2, 3), простий інтерфейс користувача (конкурент 2)	Зайняття конкурентної ніші

На основі вимог споживачів з обраних сегментів до постачальника (стартап-компанії) та до продукту, а також в залежності від обраної базової стратегії розвитку та стратегії конкурентної поведінки розробляється стратегія позиціонування, що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

Таблиця 4.17 – Визначення стратегії позиціонування

No п/ п	*Вимоги до товару цільової аудиторії*	*Базова стратегі я розвитку*	*Ключові конкурентоспроможні позиції власного стартап-проекту*	*Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)*
1.	Простота інтерфейс у, швидкість	Дифе-ренціації	Простота користувацького інтерфейсу, що дозволяє пришвидшити та спростити роботу працівників, швидкість роботи, що дозволяє підвищити швидкість експериментів	

4.4 Розробка маркетингової програми

Першим кроком є формування маркетингової концепції товару, який отримає споживач. Для цього у табл. 18 потрібно підсумувати результати попереднього аналізу конкурентоспроможності товару.

Таблиця 4.18 – Визначення ключових переваг концепції потенційного товару

No n/n	*Потреба*	*Вигода, яку пропонує товар*	*Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити*
1.	Швидкість отримання даних	ПЗ враховує реальний розподіл даних, внаслідок цього краще враховує їхні особливості і забезпечує швидший доступ до даних	Переваги у швидкості
2.	Спрощення інтерфейсу користувача	Простота роботи з ПЗ	Користувачам не потрібно замислюватись над тим, як саме обробити велики дані. Лише необхідно надати дані, та запустити програму.

Надалі розробляється трирівнева маркетингова модель товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання (табл. 19).

1-й рівень. При формуванні задуму товару вирішується питання щодо того, засобом вирішення якої потреби і / або проблеми буде даний товар, яка його основна вигода. Дане питання безпосередньо пов'язаний з формуванням технічного завдання в процесі розробки конструкторської документації на виріб.

2-й рівень. Цей рівень являє рішення того, як буде реалізований товар в реальному/ включає в себе якість, властивості, дизайн, упаковку, ціну.

3-й рівень. Товар з підкріпленням (супроводом) - додаткові послуги та переваги для споживача, що створюються на основі товару за задумом і товару в реальному виконанні (гарантії якості , доставка, умови оплати та ін). [29, 30]

Таблиця 4.19 – Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="3">Сутність та складові</td></tr>
<tr><td>І. Товар за задумом</td><td colspan="3">Об‘єкт допомогає фахівцям у області великих даних вирішувати задачу швидкого доступу до даних. Користувач має лише завантажити необхідні дані у систему та запустити обробку.</td></tr>
<tr><td rowspan="5">ІІ. Товар у реальному виконанні</td><td>Властивості/характеристики</td><td>М/Нм</td><td>Вр/Тх /Тл/Е/Ор</td></tr>
<tr><td>1. Простота інтерфейсу користувача 2. Швидкість роботи 3. Безпека згідно до світових стаднартів</td><td>-</td><td>-</td></tr>
<tr><td colspan="3">Якість: згідно до стандарту ISO 4444 буде проведено тестування</td></tr>
<tr><td colspan="3">Маркування відсутнє.</td></tr>
<tr><td colspan="3">Моя компанія. «Біг Дата – В»</td></tr>
<tr><td rowspan="2">ІІІ. Товар із підкріплю-нням</td><td colspan="3">1-місячна пробна безкоштовна версія та безкоштовне встановлення</td></tr>
<tr><td colspan="3">Постійна підтримка для користувачів</td></tr>
<tr><td colspan="4">За рахунок чого потенційний товар буде захищено від копіювання: ноу-хау.</td></tr>
</table>

Після формування маркетингової моделі товару слід особливо відмітити – чим саме проект буде захищено від копіювання. Захист може бути організовано за рахунок захисту ідеї товару (захист інтелектуальної власності), або ноу-хау, чи

комплексне поєднання властивостей і характеристик, закладене на другому та третьому рівнях товару.

Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар (остаточне визначення ціни відбувається під час фінансово-економічного аналізу проекту), яке передбачає аналіз ціни на товари-аналоги або товари субститути, а також аналіз рівня доходів цільової групи споживачів (табл. 20). Аналіз проводиться експертним методом.

Таблиця 4.20 – Визначення меж встановлення ціни

No п/п	*Рівень цін на товари-замінники*	*Рівень цін на товари-аналоги*	*Рівень доходів цільової групи споживачів*	*Верхня та нижня межі встановлення ціни на товар/послугу*
1.	25000	30000	200000	20000

Наступним кроком є визначення оптимальної системи збуту, в межах якого приймається рішення (табл. 21).

Таблиця 4.21 – Формування системи збуту

No п/п	*Специфіка закупівельної поведінки цільових клієнтів*	*Функції збуту, які має виконувати постачальник товару*	*Глибина каналу збуту*	*Оптимальна система збуту*
1.	Купують ПЗ та роблять щорічні внески для подовження ліцензії	Продаж	0(напряму), 1(через одного посередника)	Власна та через посередників

Останньою складової маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (табл. 22).

Таблиця 4.22 – Концепція маркетингових комунікацій

No п/ п	*Специфіка поведінки цільових клієнтів*	*Канали комунікацій, якими користуються цільові клієнти*	*Ключові позиції, обрані для позиціонування*	*Завдання рекламного повідомлення*	*Концепція рекламного звернення*
1.	Купівля ПЗ через інтернет, робота з ПЗ на комп‘ютерах з різними ОС	Інтернет	Швидкодія, простота використання, безпека	Показати переваги ПЗ, у тому числі і перед конкурентами	Демо-ролик із використання

4.5 Елементи фінансової підтримки стартапу та аналіз ризиків

Таблиця 4.23 – Сукупні інвестиційні витрати на реалізацію стартап-проекту

No п/п	Стаття витрат	Сукупні витрати, тис. грн.
1.	*Загальні податкові витрати*	*405*
1.1.	Проведення пошукових та прикладних досліджень	5
1.2.	Розробка проектних матеріалів і ТЕО	10
1.3.	Робоче проектування і прив'язка проекту	10
1.4.	Витрати на придбання обладнання та устаткування та пристроїв	100
1.5.	Витрати на придбання нематеріальних активів	40
1.6.	Витрати на утримання обладнання та приміщень	40
1.7.	Витрати на передвиробничі маркетингові дослідження	50
1.8.	Витрати на створення збутової мережі	50
1.9.	Витрати на просування та рекламу	50
1.10.	Оплата юридичних послуг	50
2.	*Витрати на матеріальні ресурси*	0
2.1.	матеріали	0
2.2.	комплектуючі	0
2.3.	сировина	0
3.	*Витрати на оплату праці команди стартапу*	300
Разом:		705

Таблиця 4.24 – Визначення основних фінансово-економічних показників проекту

No з/п	*Стаття витрат*	*Сукупні витрат и, тис. грн.*
1.	Обсяг виробництва продукції в натуральних показниках	1
2.	Собівартість одиниці продукції, тис. грн.	705000
3.	Собівартість виробництва продукції, тис. грн. (3 = 1 · 2)	705000
4.	Обсяг реалізації продукції в натуральних показниках	100
5.	Ціна реалізації продукції без ПДВ, тис. грн.	20000
6.	Виручка від реалізації продукції без ПДВ, тис. грн. (6 = 4 · 5)	2000000
7.	Податок на додану вартість (ПДВ), тис. грн.	200000
8.	Валовий прибуток (8 = 6 – 3)	1295000
9.	Податок на прибуток	259000
10.	Чистий прибуток (10 = 8 – 9)	1036000

Рентабельність продажів (або маржа прибутку) показує, скільки прибутку приносить кожна гривня з обсягу реалізації. Маржу прибутку, як правило, визначають окремо за кожним видом діяльності або за кожною групою реалізованої продукції за формулою:

$$Rs = \frac{\text{П}}{\text{В}} \times 100\%,$$

де П – прибуток;

В – виручка від реалізації продукції.

$$Rs = 51.8\%$$

Період окупності проекту відображає час, який потрібен для того, щоб

сума надходжень від реалізації проекту відшкодувала суму витрат на його впровадження. Період окупності звичайно вимірюється в роках або місяцях та може бути розрахований за формулою:

$$PBP = \frac{II}{ACI}$$

де PBP – період окупності інвестицій, роки;

II – сума інвестиційних витрат, тис. грн.;

ACI – щорічні надходження (річний чистий прибуток), тис. грн.

$$PBP = 0.35(\text{роки})$$

Рентабельність довгострокових інвестицій – коефіцієнт повернення інвестицій, показник рентабельності вкладень, що у відсотковому співвідношенні демонструє прибутковість (у разі, коли його значення більше 100%) або збитковість (у разі, коли його значення менше 100%) інвестицій в проект. Якщо розрахований показник менший 100%, то інвестиційні вкладення не окупуються. Показник може бути розрахований за формулою:

$$ROI = \frac{\text{Д} - \text{С}}{II} \cdot 100\%$$

де Д – дохід (виручка від реалізації продукції), тис. грн.;

С – повна собівартість, тис. грн.;

II – сума інвестиційних витрат, тис. грн.

$$ROI = 46.9\%$$

Таблиця 4.25 – Програма запобігання та реагування на ризики проекту

Ризики проекту	*Заходи запобігання ризиків*	*Заходи реагування при виникненні ризиків*
Вихід з ладу системи контролю версій	Збереження копій вихідних кодів проекту, проектної документації та інших даних на інших серверах	Отримання копії даних з інших серверів

Звільнення члена команди	Детальна декомпозиція завдань, щоб зробити кожне з них максимально простим та незалежним. Використання систем контролю версій.	Продовження роботи без цієї людини
Збільшення собівартості через проблеми роботи з бібліотеками	Немає	Необхідні додаткові час і фінансування для вирішення проблем
Зміна системного коду Tensor-Flow	Приймати участь в обговореннях переваг та недоліків цієї технології на офіційному сайті TensorFlow, абстрагувати деякі рівні ПЗ для незалежності від системного API	Змінити код у тих місцях, де використовується системне API

4.5 Висновок

Згідно до проведених досліджень існує можливість ринкової комерціалізації проекту. Також, варто відмітити, що існують перспективи впровадження з огляду на потенційні групи клієнтів, бар‘єри входження не є високими, а проект має дві значні переваги перед конкурентами.

Для успішного виконання проекту необхідно реалізувати програму із використанням засобів C/C++ та TensorFlow. В рамках даного дослідження були розраховані основні фінансово-економічні показники проекту, а також проведений менеджмент потенційних ризиків. Проаналізувавши отримані результати, можна зробити висновок, що подальша імплементація є доцільною.

ВИСНОВКИ

У даній роботі було досліджено переваги та недоліки класичних індексних структур для діапазонного пошуку, а також, в результаті цього аналізу, запропоновано і реалізовано три архітектури самоналагоджувальних індексних структур, що працюють на базі повнозв'язних нейронних мереж та методу регресії опорних векторів, також гібридної версії за допомогою підходу гурту експертів.

Перш за все, у роботі було проведено загальний огляд і аналіз класичних індексних структур для діапазонного пошуку. Загальноприйнятою індексною структурою для такого виду пошуку є B-tree. У першому розділі було розглянуто способи побудови та функціонування балансованих дерев пошуку, а також описано основні проблеми у їх використанні. Окрім цього, було розглянуто особливості побудови B-tree структур для використання у базах даних.

Так як головною проблемою роботи B-tree було визначено асимптотичний логарифмічний ріст часу пошуку з ростом кількості даних, було розглянуто можливості організації асимптотично швидшого способу доступу до даних. Для цього було показано можливості потенційної заміни CPU на GPU для організації пошуку у індексах і використання потужної паралелізації доступу до даних. З цією метою був досліджений агоритм паралелізації B-tree на GPU, який дозволяє суттєво пришвидшити час пошуку, але все одно асимптотична складність його залишає бажати кращого — потрібна організація пошуку за константний час. Тому було запропоновано альтернативні рішення, що базуються на використанні підходів машинного навчання у якості побудови предиктивних моделей.

Було показано, що для досягнення асимптотично константного пошуку в якості моделі діапазонного пошуку доцільно використовувати функцію розподілу даних. Також було також показано, що для апроксимації цієї функцію варто використовувати моделі машинного навчання.

Було проведено ряд експериментів з вивчення розподілу даних. Для цього було підібрано два типи наборів даних: числові та строкові.

При виборі моделей машинного навчання, варто було враховувати, що вони мають працювати доволі швидко. Внаслідок цього, було запропоновано використання неглибоких нейронних мереж та методу регресії опорних векторів, адже такі моделі є простими та ефективними.

Було реалізовано самоналагоджувальні індексні структури для діапазонного пошуку, що базуються на використанні зазначених моделей машинного навчання. Для реалізації цих індексних структур були використані сучасні засоби і середовища програмування та бібліотеки для машинного навчання, що значно полегшують процес тренування та використання моделей.

Після цього були проведені експерименти з реалізованими індексними структурами на двох наборах даних. Спочатку було реалізовано індексні моделі на базі використання неглибоких нейронних мереж та на базі використання методу регресії опорних векторів. Варто зазначити, що результати експериментів показали наявність проблеми передбачення на “останній милі”, як було зазначено у попередніх досідженнях. Іншими словами, модель гарно вивчає загальний розподіл даних, але може доволі сильно помилятись при визначенні точної позиції. Для вирішення проблеми були застосовані моделі типу гурт експертів. Для цього класу було зроблено порівняння моделей передбачення на останній милі у вигляді нейронних мереж та методу регресії опорних векторів, які показали схожі результати в залежності від набору даних. Автоматичний підбір гіперпараметрів на основі методів Байєсівської оптимізації дозволив покращити результати роботи моделі гурту експертів, яка показала загалом найкращий результат.

Було наведено результати експериментів по використанню запропонованих самоналагоджувальних індесних структур на різних наборах даних: чисельних та строкових. Ці результати показали придатність використання навчених індексних структур та перспективу їх подальшого дослідження та інтеграції у бази даних або сторонні системи індексації.

Внаслідок того, що розроблені самоналагоджувальні індексні структури для діапазонного пошуку показують більш ефективне використання пам’яті на деяких наборах даних, їх можна інтегрувати у сучасні системи керування базами даних, за умови подальшого вдосконалення.

Важливо також зауважити, що результати, отримані в цій роботі, підтверджують загальну ідею доцільності використання машинного навчання для заміни класичних індексних структур.

ПЕРЕЛІК ПОСИЛАНЬ

1. The Zettabyte Era: Trends and Analysis [Електронний ресурс] – Режим доступу до ресурсу: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html.
2. T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The Case for Learned Index Structures [Електронний ресурс] – Режим доступу до ресурсу: https://arxiv.org/abs/1712.01208.
3. Moore Law is Dead but GPU will get 1000X faster by 2025 [Електронний ресурс] – Режим доступу до ресурсу: https://www.nextbigfuture.com/2017/06/moore-law-is-dead-but-gpu-will-get-1000x-faster-by-2025.html.
4. Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom. Database Systems, The Complete Book, Second Edition / Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom. – Pearson Prentice Hall, 2009.
5. Comer, Douglas. The Ubiquitous B-Tree. / Comer, Douglas - Computing Surveys, 1979 11 (2): 123–137.
6. Peter Bakkum, Kevin Skadron. Accelerating SQL Database Operations on a GPU with CUDA. In GPGPU '10: Proceedings of the Third Workshop / Peter Bakkum, Kevin Skadron – GeneralPurpose Computation on Graphics Processing Units, New York, NY, USA, 2010, c. 94-103.
7. Jordan Fix, Andrew Wilkes, and Kevin Skadron. Accelerating Braided B+ Tree Searches on a GPU with CUDA / Jordan Fix, Andrew Wilkes, and Kevin Skadron – Proceedings of the 2nd Workshop on Applications for Multi and Many Core Processors: Analysis, Implementation, and Performance (A4MMC), June 2011.
8. Krzysztof Kaczmarski, Experimental B+-tree for GPU / Krzysztof Kaczmarski - Proc. of ADBIS, vol. 2, Rome, Italy, 2011, pp. 232–241.
9. Kevin P. Murphy, Machine Learning: A Probabilistic Perspective / Kevin P. Murphy – The MIT Press, 2012, c. 9-10.
10. Gershenson, Carlos, Artificial Neural Networks for Beginners. 2003.

11. NeuralNet.info Учебник - Режим доступу до ресурсу: https://neuralnet.info/chapter/%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-%D0%B8%D0%BD%D1%81/#%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F-%D0%B0%D0%BA%D1%82%D0%B8%D0%B2%D0%B0%D1%86%D0%B8%D0%B8
12. Xavier Glorot, Antoine Bordes and Yoshua Bengio. Deep sparse rectifier neural networks. / Xavier Glorot, Antoine Bordes and Yoshua Bengio – AISTATS, 2011.
13. Yann LeCun, Leon Bottou, Genevieve B. Orr and Klaus-Robert Müller, Efficient BackProp. / Yann LeCun, Leon Bottou, Genevieve B. Orr and Klaus-Robert Müller – G. Orr and K. Müller. Neural Networks: Tricks of the Trade. Springer, 1998.
14. Vladimir Vapnik, The Nature of Statistical Learning Theory. / Vladimir Vapnik – Springer, New York, 1995
15. Mathematical Formulation of SVM Regression — режим доступу до ресурсу: https://www.mathworks.com/help/stats/understanding-support-vector-machine-regression.html
16. Jasper Snoek, Hugo Larochelle, Ryan P. Adams, Practical Bayesian optimization of machine learning algorithms / Jasper Snoek, Hugo Larochelle, Ryan P. Adams – Advances in Neural Information Processing Systems, 2012, с. 2951-2959.
17. Документація Jupyter — Режим доступу до ресурсу: https://jupyter.readthedocs.io/en/latest/
18. Фреймворк Keras - Режим доступу до ресурсу: https://keras.io/
19. Фреймворк Tensorflow — Режим доступу до ресурсу: https://www.tensorflow.org/
20. Бібліотека NumPy — Режим доступу до ресурсу: http://www.numpy.org/
21. Pytorch | About – Режим доступу до ресурсу: https://pytorch.org/about/
22. Scikit-learn – Режим доступу до ресурсу: http://scikit-learn.org/
23. Пакет Bayesian Optimization – Режим доступу до ресурсу: https://github.com/fmfn/BayesianOptimization
24. Blog Safecast — Режим доступу до ресурсу: https://blog.safecast.org/data/

25. Tara Baldacchino, Elizabeth J. Cross; Keith Worden, Jennifer Rowson, Variational Bayesian mixture of experts models and sensitivity analysis for nonlinear dynamical systems / Tara Baldacchino, Elizabeth J. Cross; Keith Worden, Jennifer Rowson – Mechanical Systems and Signal Processing, 2016, с. 66-67.
26. Прогнозування ефективності інвестиційного проекту — Режим доступу до ресурсу: http://pidruchniki.com/1566072162240/turizm/prognozuvannya_efektivnosti_investitsiynogo_proektu
27. Стратегічний аналіз підприємства — Режим доступу до ресурсу: http://pidruchniki.com/1974070454048/menedzhment/strategichniy_analiz_pidpriyemstva
28. Стратегічний аналіз підприємства — Режим доступу до ресурсу: http://pidruchniki.com/1974070454048/menedzhment/strategichniy_analiz_pidpriyemstva
29. Трехуровневая модель товара — Режим доступу до ресурсу: http://pidruchniki.com/19670511/marketing/razrabotka_produktovoy_strategii_kompanii#891
30. Маркетингове поняття та класифікація видів товару – Режим доступу до ресурсу: http://marketing-helping.com/konspekti-lekcz/21-konspekt-lekczj-qosnovi-marketinguq/412-marketingove-ponyattya-ta-klasifkaczya-vidv-tovaru.html